A framework for evaluating unstructured text data using sentiment analysis

Jacqueline Kazmaier

Dissertation presented for the degree of
Doctor of Philosophy in Industrial Engineering
in the Faculty of Engineering at Stellenbosch University

Supervisor: Professor JH van Vuuren

December 2020

Declaration

By submitting this dissertation electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: December 1, 2020

Abstract

Increased exposure of the average customer and citizen to polarised content from various sources has been a cause of significant concern for companies and governmental organisations. Such content has, for example, served as a catalyst for violent uprisings and shifts in stock market prices. The collection and study of opinion have therefore become a necessity in many industries. Due to the vast extent of such data, however, manual approaches to this end are no longer feasible. This situation has given rise to the research field of *sentiment analysis* or *opinion mining* — the *computational* study of people's opinions, attitudes and emotions.

Whereas the task of sentiment parsing is relatively easy for humans, the subtle nuances of natural languages render this task inherently difficult for computers. This is especially true in the South African context, where opinion-bearing expressions may be composed in up to eleven different languages. Automated sentiment analysis tools developed in one setting are, therefore, often ineffective in another. Furthermore, an abundance of research has been dedicated to developing algorithms for the purpose of classifying sentiment, while little guidance exists on how to incorporate this information into the decision-making processes of affected entities.

In this dissertation, a generic framework for sentiment analysis is proposed, with a focus on facilitating the model development process for a user in a manner such that good performance may be achieved irrespective of the problem domain. The objective of the framework is ultimately to facilitate a flexible, exploratory analysis of model results in combination with existing structured attributes in order to gain actionable insights. The framework may aid organisations in successfully leveraging unstructured, opinion-bearing data in combination with structured data sources with a view to inform effective decision making.

An instantiation of this framework is implemented on a computer as a concept demonstration. This implementation is applied to a real-world case study in the South African banking sector in order to illustrate the practical applicability of the framework. Furthermore, the framework's ability to generalise across domains is validated by means of three additional case studies in respect of freely available benchmark data. During this process, the models developed by means of the framework are shown to be competitive with published benchmark results. Moreover, the framework is shown to address and successfully overcome shortcomings of existing frameworks in the literature.

Opsomming

Toenemende blootstelling van die gemiddelde kliënt en burger aan gepolariseerde inhoud uit verskeie bronne het 'n groot bron van kommer vir ondernemings en regeringsorganisasies geword. Sulke inhoud het byvoorbeeld al as katalisators vir gewelddadige opstande en veranderinge in aandeelpryse gedien. Die insameling en bestudering van menings het dus in baie nywerhede 'n noodsaaklikheid geword. Vanweë die groot omvang van sulke data, is handmatige benaderings daartoe egter nie meer uitvoerbaar nie. Hierdie situasie het aanleiding gegee tot die navorsingsveld van *sentimentanalise* of *meningsontginning* — die *berekeningstudie* van mense se opinies, houdings en emosies.

Terwyl die taak van sentiment-ontleding relatief maklik is vir mense, maak die subtiele nuanses van natuurlike tale hierdie taak inherent moeilik vir rekenaars. Dit is veral waar in die Suid-Afrikaanse konteks, waar meningsdraende uitdrukkings in tot elf verskillende tale geformuleer kan word. Instrumente vir outomatiese sentiment-analise wat in een omgewing ontwikkel is, is dus dikwels nie noodwendig in 'n ander omgewing doeltreffend nie. Verder is 'n oorvloed navorsing aan die ontwikkeling van algoritmes vir sentiment-klassifikasie toegewy, terwyl daar weinig riglyne bestaan oor hoe om hierdie inligting in die besluitnemingsprosesse van belanghebbende entiteite te inkorporeer.

In hierdie proefskrif word 'n generiese raamwerk vir sentimentanalise daargestel, met die klem op fasilitering van die modelontwikkelingsproses op só 'n manier dat goeie werkverrigting, ongeag die probleemdomein, bereik kan word. Die uiteindelike doel van die raamwerk is die fasilitering van 'n buigsame, verkennende analise van modelresultate in kombinasie met bestaande gestruktureerde eienskappe ten einde insigte te verwerf wat na sinvolle aksies kan lei. Die raamwerk kan organisasies help om ongestruktureerde, meningsvormende data suksesvol in kombinasie met gestruktureerde databronne te benut met die oog op doeltreffende besluitneming.

'n Spesiale geval van hierdie raamwerk word rekenaarmatig as 'n konsepdemonstrasie geïmplementeer. Hierdie implementasie word op 'n werklike gevallestudie in die Suid-Afrikaanse banksektor toegepas om die praktiese toepasbaarheid van die raamwerk te illustreer. Verder word die raamwerk se vermoë om oor verskeie terreine te veralgemeen deur middel van drie addisionele gevallestudies in die konteks van vrylik beskikbare maatstafdata bekragtig. Gedurende hierdie proses word daar bevind dat die modelle wat deur middel van die raamwerk ontwikkel is, met gepubliseerde maatstafresultate mededingend blyk te wees. Verder word daar getoon dat die raamwerk tekortkominge van bestaande raamwerke in die literatuur aanspreek en suksesvol oorkom.

Acknowledgements

The author wishes to acknowledge the following people and institutions for their various contributions towards the completion of this work:

- My supervisor and mentor, Prof Jan van Vuuren, for his selfless devotion to helping every one of his students to achieve their full academic and personal potential, for his guidance, support and critique, and for opening my mind to a vast range of new ideas. You have shaped a great part of my life and I am immeasurably grateful to have had the opportunity to walk a part of the journey with you.
- The *Stellenbosch Unit for Operations Research in Engineering* (SUnORE) and its members for creating a working environment that is not only academically stimulating, but has also helped me to further my personal development, expand my world view and forge new friendships. I will forever treasure the time I spent here.
- The Department of Industrial Engineering at Stellenbosch University for the use of its facilities, and for fostering a familiar atmosphere in which every student feels valued.
- The Harry Crossley Foundation, the Ernst and Ethel Eriksen Trust, SUnORE and the Department of Industrial Engineering for their generous financial support.
- Stellenbosch University International, the Department of Informatics and Mathematics at the Technical University of Munich and the Erasmus Foundation for giving me the opportunity to acquire the necessary theoretical foundations for this dissertation during an international exchange. My experience in Munich was invaluable from both an academic and a personal perspective.
- The industry partner (who prefers to remain anonymous) for providing data for analysis in this dissertation and offering technical advice.
- My wonderful parents, Sylvia Kaschik and Dave Morris, as well as Peter and Birgit Kazmaier, for their unconditional support and encouragement throughout the course of my studies. Thank you for always believing in me and for giving me the opportunity to further my academic career. I carry a part of each of you in me.
- My sister, Angelina Kazmaier, for celebrating my smallest successes with profound enthusiasm and for being an endless source of encouragement. I appreciate you dearly.
- Last, but not least, my partner and best friend, Bernd Rohrmüller, for his unwavering support, inspiration and patience. Thank you for engaging in conversations about the sentiments of strangers and the inner workings of machine learning algorithms at all hours of the day and for continually trying to help me to shorten my response to the question what my thesis is about. You make everything better.

Table of Contents

List of Acronyms

Acronym	Description	Acronym	Description
ADAM	Adaptive Movement Estimation	**LDA**	Latent Dirichlet Allocation
AI	Artificial Intelligence	**LSA**	Latent Semantic Analysis
ANN	Artificial Neural Network	**LSI**	Latent Semantic Indexing
AUC	Area Under the Curve	**LSTM**	Long Short-Term Memory
ATM	Automatic Teller Machine	**MAP**	Maximum A Posteriori
BBL	Best Base Learner	**MLE**	Maximum Likelihood Estimation
BL	Base Learner	**MST**	Model Selection Triple
BFGS	Broyden-Fletcher-Goldfarb-Shanno	**NLTK**	Natural Language Toolkit
CA	Classification Accuracy	**OLAP**	OnLine Analytical Processing
CART	Classification And Regression Trees	**PCA**	Principal Component Analysis
CBOW	Continuous Bag-Of-Words	**PMI**	Point-wise Mutual Information
CG	Conjugate Gradient	**POS**	Parts-Of-Speech
CNN	Convolutional Neural Network	**PR**	Public Relations
CRM	Customer Relationship Management	**ReLU**	Rectified Linear Unit
CSV	Comma Separated Values	**RMSProp**	Root-Mean-Squared Propagation
DFD	Data Flow Diagram	**RNN**	Recurrent Neural Network
DM	Discrete output with Meta-learning	**ROC**	Receiver Operating Characteristic
DS	Discrete output with Simple voting	**SAG**	Stochastic Average Gradient
DSS	Decision Support System	**SDLC**	Systems Development Life Cycle
DBMS	Data Base Management System	**SGD**	Stochastic Gradient Descent
DW	Discrete output with Weighted voting	**SM**	Scoring output with Meta-learning
ECCO	Evaluating a Corpus Characterised by Opinion-bearing language	**SMS**	Short Message Service
EDA	Exploratory Data Analysis	**SMO**	Sequential Minimal Optimisation
ERD	Entity Relationship Diagram	**SO-A**	Semantic Orientation from Association
FN	False Negatives	**SS**	Scoring output with Simple voting
FP	False Positives	**SVD**	Singular Value Decomposition
GPU	Graphics Processing Unit	**SVM**	Support Vector Machine
GRU	Gated Recurrent Unit	**SW**	Scoring output with Weighted voting
GUI	Graphical User Interface	**tanh**	Hyperbolic tangent
IDE	Integrated Development Environment	**TF-IDF**	Term Frequency-Inverse Document Frequency
IR	Information Retrieval	**TN**	True Negatives
IS	Information System	**TOM**	Twitter Opinion Mining
JSON	JavaScript Object Notation	**TP**	True Positives
kNN	k-Nearest Neighbours	**UML**	Unified Modelling Language
KT	Kuhn-Tucker	**US**	United States

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

Contents

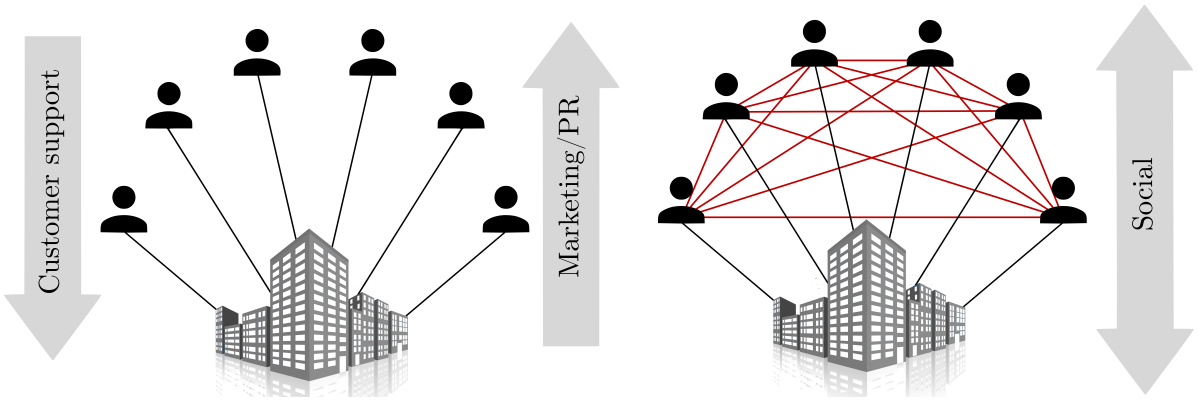
1.1 Background

"With public sentiment, nothing can fail. Without it, nothing can succeed." — Abraham Lincoln

The opinion of others has influenced the human decision-making process for decades. This is particularly prevalent when a choice involves expending valuable resources such as time and money, in which case people often rely on the past experiences of their peers [43].

With the explosive growth of the Internet and social media, this has become an increasingly observed phenomenon. As the importance of the Internet as a source of information has grown to exceed that of traditional sources of knowledge, it has also become a platform for sharing ideas and experiences. It is now possible to draw on the opinions of a vast pool of people, consisting not only of acquaintances and professional critics, but also of complete strangers posting their opinions in the public domain [231].

One of the consequences of this development for the business sector is the powerful influence that past and current customers now have on each other and on potential future customers. The effect of social media on the communication channels between companies and their customers is illustrated in Figure 1.1. Wu [344] explained three key differences between the traditional communication model in Figure 1.1(a) and the modern communication model in Figure 1.1(b). First, traditional communication channels are *transient* and *opaque*: Communications between client and company are often not recorded and, even when they are, these records are typically not retrievable by customers. With the introduction of social media, interactions are now public and available for viewing by current and potential customers for an indefinite period of time. They are therefore *transparent* and *persistent*.



(a) *Traditional communication model* (b) *Modern communication model*

FIGURE 1.1: *A schematic comparison of the traditional and modern models of communication between an organisation and its customers (adapted from Wu [344]).*

Secondly, customer-to-company communication has traditionally been managed by customer support departments, whilst marketing and public relations departments have been responsible for company-to-customer communication. In both cases, the communication is *unidirectional*. Communication *via* social media channels, on the other hand, is *bi-directional* since both parties are able to create content on the platform.

Lastly, since the customers in the traditional communication model are unaware of the identities and particulars of other customers, the model may be described as *one-to-many*. This property emerges due to the transience and opacity of communication, and is reflected in the single point of origin of all lines in Figure 1.1(a). In the modern model, however, coordination between customers is also possible. This model therefore describes a *many-to-many* relationship.

The effect of inter-customer communication is significant. A market study commissioned by *Popimedia*[1] in 2017 indicated that 78% of South African consumers consult online product reviews before making an in-store purchase [292]. Another survey indicated that 93% of consumers in the *United States* (US) read local reviews in order to judge a business [38]. According to a *Google* consumer study of over 1 000 participants, 67% of consumers are influenced by such reviews. For instance, it was found that 22% of potential customers will refrain from purchasing a product after reading merely one negative review, whilst 59% will do so after encountering three negative reviews [123]. Furthermore, 48% of participants in the US survey reported being unwilling to do business with companies that have an online rating of less than four out of five stars [38].

Effectively managing client feedback may, on the other hand, have a positive influence on business performance. The latter study also found that 73% of customers trust a local business more after reading a positive review [38]. Moreover, it has been shown that the inclusion of customer reviews on a company website leads to an increase in revenue per site visit [290]. Merely enabling reviews is, however, not sufficient — 30% of survey respondents listed whether or not the business had responded to a review as a critical factor when judging a business based on reviews [38].

A problem arises, therefore, due to the sheer volume of data that have to be processed for the purposes of studying and addressing customer feedback. Changes in technology and culture have made it both easier and more common for customers to comment on an organisation's products and services. Furthermore, for every n customers and corresponding company-to-customer

[1] *Popimedia* is a global advertising technology platform and the largest Facebook marketing partner in Africa.

communication channels in Wu's modern model in Figure 1.1(b), there are $\binom{n}{2}$ possibilities for customer-to-customer communication. This is illustrated in the figure using black-coloured and red-coloured communication lines, respectively. In the 6-customer example shown, there are 15 red (customer-to-customer) lines, compared with only 6 black (company-to-customer) lines. If the number of customers is increased to 1 000, for instance, there exist 499 500 possibilities for inter-customer communication that must be monitored.

In line with Wu's argument [344], it is therefore not feasible to increase the number of employees responsible for monitoring customer feedback at the same rate as the number of communication channels. Instead, it is suggested that customer relationships must be managed using *customer relationship management* (CRM) systems which employ automated methods. This necessitates changes in the technology, as well as in the business processes and company culture, adopted by an organisation.

From a technological perspective, CRM systems perform two primary functions. First, they must interpret and understand unstructured text data. Secondly, since employees of the organisation are unable to engage in every customer conversation, these systems must prioritise relevant conversations and direct these to the appropriate responder or business process [344]. The design of the latter function is shaped by requirements and conditions specific to the organisation in question. The former function calls upon the field of *sentiment analysis* or *opinion mining*, which is concerned with "the computational treatment of opinion, sentiment, and subjectivity in text" [231].

A prevalent task in sentiment analysis is the classification of an observation into one of various categories (*e.g. positive* and *negative*, or *happy* and *sad*). There are two fundamental approaches to sentiment classification, namely *lexicon-based* and *computational learning* approaches [93, 111]. The former approach relies on a large-scale knowledge base or publicly available lexicons, and classifies the sentiment of a given text based on predefined polarities of the words or phrases contained within the text. The latter approach, on the other hand, attempts to learn patterns from input data labelled with their semantic orientation in order to classify new and unlabelled data based on these patterns.

The potential applications of the field of sentiment analysis are vast and powerful. Shifts in sentiment on social media have, for example, been shown to correlate with shifts in the stock market [19]. In the political domain, public sentiment has been used as a predictor of election results. Opinionated postings on social media have, furthermore, led to events of mass activism such as the Egyptian Revolution in 2010 and other uprisings during the Arab Spring of 2010–2012 [176, 325]. The collection and study of opinion, using both external data from the Internet and internal data, such as direct customer communication, have therefore become a necessity in many industries [176].

1.2 Informal problem description

Whilst there is an abundance of research dedicated to developing algorithms for the purpose of classifying sentiment, little guidance exists on how to apply these algorithms in practice, as well as how to incorporate model results into the decision-making process of affected entities.

Existing frameworks for sentiment analysis in the literature focus on the implementation of a *specific* model with a predetermined set of features. The fact that the classification performance of such a model depends on the domain in which it is applied, as well as the feature set employed as input [280, 331], is typically not addressed by these frameworks. Furthermore, the process

of tuning the parameters of machine learning models, in particular, is rarely included in such frameworks in spite it being an essential part of the model development process [165].

Moreover, only few frameworks go beyond simple summaries of the proportional representation of specified sentiment classes as determined by the results of a given classification model. In order to effectively leverage sentiment analysis in an organisation, a deeper analysis of the model results is required, focusing on the topics discussed in reviews and the relationship between sentiment and various additional information that may be available.

The aim in this dissertation is to design a generic framework for the evaluation of unstructured, opinion-bearing text data that addresses these shortcomings. This framework should facilitate the process of preparing the data for analysis, extracting and selecting features from the unstructured text, as well as evaluating and selecting (or combining) suitable models used for classifying sentiment. Furthermore, the analysis and synthesis of the results of selected models should be accommodated, with the aim of extracting patterns and information from the data and presenting these to the user in a meaningful way. The framework may then be integrated into a *decision support system* (DSS), a concept demonstration of which is showcased in this dissertation.

1.3 Scope and objectives

The following objectives are pursued in this dissertation:

I To *conduct* a thorough survey of the literature related to:

  (a) The development and evaluation of machine learning models for classification problems,

  (b) The analysis of unstructured data, with a particular focus on the transformation of text data into a structured format,

  (c) The sentiment analysis problem and its common solution approaches, with a particular focus on the use of machine learning algorithms, and

  (d) The design and development of decision support systems.

II To *design*, based on the literature review of Objective I, of a generic DSS framework which may be used to evaluate opinion-bearing data in the form of unstructured text. This framework should facilitate:

  (a) The preprocessing of unstructured text data,

  (b) The transformation of preprocessed text into a structured format for use by a sentiment classification model,

  (c) The development, evaluation, comparison and combination of suitable sentiment classification models, and

  (d) The analysis of model results with the objective of extracting actionable insight.

III To *implement* an instantiation of the framework of Objective II in an applicable software platform.

IV To *verify* and validate the implementation of Objective III according to generally accepted modelling and DSS design guidelines.

V To *apply* the proof-of-concept implementation of Objective III to a real-world case study in order to demonstrate its practical application.

VI To *validate* the general applicability of the framework of Objective II by means of further case studies from various domains.

VII To *recommend* sensible follow-up work related to the work in this dissertation which may be pursued in future.

The case study of Objective V is limited in scope to the design, implementation, testing and validation of a concept demonstrator of the proposed DSS. The development and integration of a full-scale DSS with the business processes and database of the case study partner organisation is not pursued. Furthermore, the sentiment analysis framework of Objective II is designed to facilitate the sentiment classification of text at the *document level*, focusing particularly on the use of machine learning models.

1.4 Research methodology

The research underlying this dissertation is executed in five phases. The first phase entails consulting the existing academic literature in pursuit of Objective I. The necessary theoretical foundation of the work in this dissertation related to the field of machine learning, particularly in the context of classification problems, is established in Objective I(a). A fundamental understanding of data analysis is pursued in Objective I(b), with a particular focus on the process followed to analyse data of an unstructured nature. In pursuit of Objective I(c), the more specific problem of sentiment analysis is explored. Attention is afforded to the formulation of the problem of sentiment classification, following which a survey of existing solution approaches in the literature and their relative effectiveness is conducted. Furthermore, existing approaches for sentiment summarisation are also investigated. The methods identified in this survey serve as a guideline and justification for the framework instantiation put forward in pursuit of Objective III. Finally, best practices in respect of the design of DSSs are studied in fulfilment of Objective I(d) and in support of Objectives II–VI. This phase also encompasses the acquisition of relevant technical skills by developing a proficiency in the `Python` programming language with a particular focus on its machine learning and natural language processing libraries.

The second phase of the research constitutes the development of a generic framework for evaluating unstructured data by means of sentiment analysis in pursuit of Objective II. The theoretical knowledge acquired during the first phase described above is employed to guide the design of the framework, so as to ensure that state-of-the art techniques are employed for each of the tasks it facilitates and that its generic nature is preserved. As such, the framework is designed to be modular and to allow for the application of various different algorithms and solution approaches, as well as the extension and replacement of modules.

The third phase is executed in pursuit of Objective III and comprises the iterative development of a *specific* instantiation of the *generic* framework designed during the second phase. Certain algorithms and parameters are thus chosen to populate the framework for the purpose of implementing the processes described therein in the form of a computer program, which may be used to demonstrate the working of the framework. This phase also involves the verification and validation of this proof-of-concept implementation in pursuit of Objective IV according to the guidelines studied during the literature review.

The fourth phase of the study comprises the application of the proof-of-concept implementation to real-world case studies. This is implemented in two stages. During the first stage, a detailed

case study is conducted in respect of data from an industry partner in pursuit of Objective V. By demonstrating how each of the processes of the framework may be applied practically to preprocess text data, generate and compare suitable models for sentiment classification, and extract information and actionable insight from the results of the selected classification model, the value of the framework is illustrated. During the second stage, several additional case studies are conducted in respect of data from various domains in pursuit of Objective VI. These data are selected to cover a range of application areas and data characteristics, in order to demonstrate the framework's ability to generalise across domains and problem types.

The final phase of the research entails the appraisal of the contributions of this dissertation in order to suggest suitable future research endeavours.

1.5 Dissertation organisation

Apart from this introductory chapter, this dissertation comprises a further twelve chapters (organised in five parts), a bibliography and five appendices. Part I of the dissertation is a literature review and consists of four chapters, Chapters 2–5. Chapter 2 is devoted to presenting the reader with prerequisite mathematical and statistical material related to this dissertation. The chapter opens with a description of relevant statistical distributions. Important concepts in matrix algebra are then reviewed, and this is followed by a description of selected statistical models for data analysis. The chapter closes with a review of selected algorithms for solving non-linear optimisation problems.

Chapter 3 contains an overview of the field of machine learning. Various *types* of machine learning are first distinguished, and this is followed by an outline of the typical training procedure and evaluation metrics employed in the case of classification models. Subsequently, selected machine learning algorithms pertaining to this dissertation are described.

Chapter 4 contains a survey of the literature related to the analysis of unstructured data and the field of sentiment analysis. The process typically followed to analyse data of an unstructured nature is first described, and this is followed by an account of several techniques suitable for processing such data. A brief outline of the history of sentiment analysis is then given, and this is followed by an account of the various tasks and levels of analysis that prevail in this area. Subsequently, common techniques for analysing and synthesising sentiment are described.

The final chapter of Part I of this dissertation, Chapter 5, relates to DSSs. The chapter opens with a description of the primary components of a typical DSS. Various types of DSSs are then distinguished. Finally, popular methodologies for developing, verifying and validating such systems are outlined.

Part II of the dissertation consists of two chapters, Chapters 6 and 7, and focuses on the proposed sentiment analysis framework. In Chapter 6, this framework is presented. The chapter, however, opens with an account of similar existing frameworks from the literature. Subsequently, a generic data science paradigm is described within which the proposed framework is set. Finally, a high-level overview of the proposed framework is given, and this is followed by a detailed description of its major components by means of data flow diagrams.

The proof of concept implementation of the framework is demonstrated in Chapter 7. An overview is first given of the development of the implementation in the form of a computerised system, including the design choices made during its development. Thereafter, the system is described in detail in terms of the framework's primary components, and the specific algorithms or processes chosen to populate each respective component of the framework are outlined. The

chapter closes with an account of the procedures followed in order to verify and validate the system.

A real-world case study to which the implementation of the framework is applied is the focus of Part III of the dissertation, which consists of two chapters, Chapters 8 and 9. In Chapter 8, a background to the case study is given, as well as a detailed description of the data associated with the study and any data preparation that was conducted prior to the study. Finally, the objectives pursued during the execution of the case study are outlined.

The analysis results pertaining to the case study are presented in Chapter 9. The manner in which the case study data were preprocessed is first described. This is followed by a detailed description of how various sentiment classification models were developed during the course of the computerised implementation of the framework. Finally, the results returned by the best-performing model are analysed as per the process outlined in the framework in order to extract valuable information and insight related to the case study.

Part IV of the dissertation is concerned with the validation of the framework by means of further case studies. The first chapter in this part, Chapter 10, contains descriptions of each of the benchmark data sets selected for this purpose, as well as an account of any data preparation activities performed in respect of these data. The associated validation analyses are described in the subsequent chapter, Chapter 11. These analyses serve to evaluate the framework's ability to generalise across domains and to compare the classification performance of the sentiment models generated by means of the framework with the results of other researchers.

The final part of the dissertation, Part V, consists of two chapters, Chapters 12 and 13. Chapter 12 contains a summary of and reflection on the contributions of the dissertation. A number of ideas are finally provided in Chapter 13 for possible future follow-up work building upon the contributions of this dissertation.

Part I

Literature review

CHAPTER 2

Mathematical and statistical preliminaries

Contents

The purpose of this chapter is to present the reader with prerequisite mathematical and statistical material for the understanding of machine learning and sentiment analysis techniques. Statistical distributions that are relevant in this context are first reviewed, and this is followed by a review of important concepts in matrix algebra. Subsequently, selected statistical models for data analysis are described. Finally, an account is given of selected algorithms for solving non-linear optimisation problems.

2.1 Relevant statistical distributions

The act of rolling a die or of measuring the ambient temperature may be described as an *experiment*. In rare cases, experiments can be conducted in a *controlled* setting, in the sense that all factors influencing the outcome of the experiment can be manipulated. In most *real-world* scenarios, however, external factors influence the experiment that are beyond the control of the conductor. Consequently, a different experimental outcome may be observed if an experiment is replicated in the same manner. Such experiments are referred to as *random experiments*.

The outcome of an experiment is often summarised by a single number (*e.g.* the number of eyes on the top face of the die after it has been rolled). A *random variable* X is a function by which a number is assigned to each possible outcome of a random experiment. Such a variable may be *discrete*, if it has a finite (or countably infinite) range, or *continuous*, if its range is an interval of real numbers [16, 200].

Finally, each possible outcome of a random experiment may occur with a certain *probability*. One may then define a *probability distribution* of a random variable X as a description of the probabilities associated with each of the possible values of X. This may be expressed simply as a list, or by means of a *probability mass function* $f(x)$ which satisfies the following conditions for a discrete random variable:

(i) $f(x_i) \geq 0$,

(ii) $\sum_{i=1}^{n} f(x_i) = 1$, and

(iii) $f(x_i) = P(X = x_i)$,

where $x_1, x_2, \ldots, x_n$ are the possible values of X and $P(X = x_i)$ is the probability of outcome x_i being observed in the experiment [200]. The probability distribution may be summarised by two numbers, namely the *mean*

$$\mu = \sum_{x} x f(x)$$

and the *variance* or *second central moment*

$$\sigma^2 = \sum_{x} (x - \mu)^2 f(x),$$

which represent the centre and the level of dispersion or variability of the distribution, respectively. The definitions for continuous distributions are similar, with the sum operator replaced by an integral [16, 200]. In the following sections, probability distributions pertinent to the analyses carried out later in this dissertation are introduced.

2.1.1 The Bernoulli and binomial distributions

Consider a random experiment with only two possible outcomes, $x_1 = 1$ and $x_2 = 0$. Such an experiment is commonly referred to as a *Bernoulli trial* [200]. The distribuion of X may, in this case, be represented as

$$P(X = x_1) = 1 - P(X = x_2) = p,$$

or, alternatively, as

$$f(x) = p^x (1 - p)^{1-x},$$

where $0 < p < 1$. This is known as the *Bernoulli distribution* [16, 208]. Now consider a random

experiment, which consists of n Bernoulli trials such that:

(i) The trials are independent,

(ii) each trial results in only two possible outcomes, denoted here as *success* and *failure*, and

(iii) the probability of a success in each trial, p, remains constant.

Suppose the random variable X represents the number of successes observed in the experiment and has parameters $n \in \mathbb{N}$ and $0 < p < 1$. It can be shown that X is distributed according to the *binomial distribution* [200, 342], given by

$$P(X = m) = \binom{n}{m} p^m (1-p)^{n-m}, \quad m \in \{0, 1, \ldots, n\}.$$

This is often expressed using the notation $X \sim B(n, p)$. A plot of the binomial distribution for $n = 20$ and $p = 0.5$ is shown in Figure 2.1.

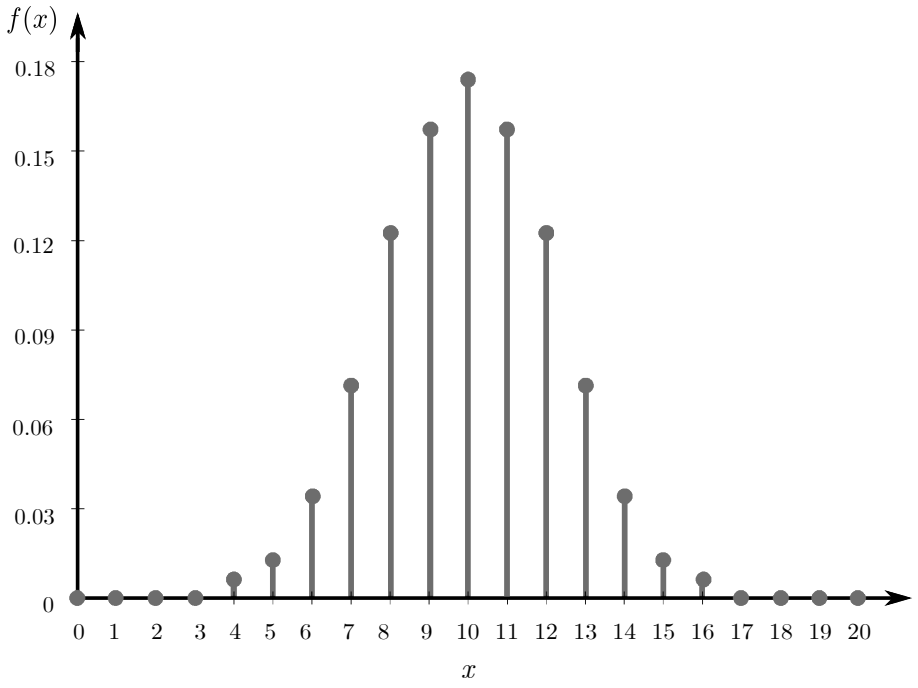


FIGURE 2.1: *Plot of a binomial distribution for $n = 20$ and $p = 0.5$ (adapted from Montgomery and Runger [200]).*

2.1.2 The multinomial distribution

The binomial distribution can be generalised to model the *joint distribution* of multiple discrete random variables. In this case, an experiment consists of n trials and it is assumed that:

(i) The trials are independent,

(ii) the result of each trial can be categorised into one of k classes, and

(iii) the probability of the result of a trial belonging to class 1, ..., class k is constant for all trials and is equal to $p_1, \ldots, p_k$, respectively.

Suppose the random variables $X_1, \ldots, X_k$ represent the number of trials that result in class 1, class 2, ..., class k outcomes, respectively. These variables follow a *multinomial distribution* with the joint probability mass function

$$P(X_1 = x_1, \ldots, X_k = x_k) = \frac{n!}{x_1! \cdots x_k!} p_1^{x_1} \ldots p_k^{x_k}$$

for $x_1 + \ldots + x_k = n$ and $p_1 + \ldots + p_k = 1$ [30, 200].

2.1.3 The Gaussian distribution

The most widely used continuous distribution is the *Gaussian* or *normal distribution*. According to the *central limit theorem*, the distribution of the mean (or sum) of n independent random variables with arbitrary distributions approaches a normal distribution as n becomes large [342].

The Gaussian distribution of a random variable X with mean μ and variance σ^2 is given by the probability density function

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(x-\mu)^2}{2\sigma^2}}, \quad x \in \mathbb{R},$$

which has a distinct *bell curve* graph, such as the one shown in Figure 2.2 [200, 342].

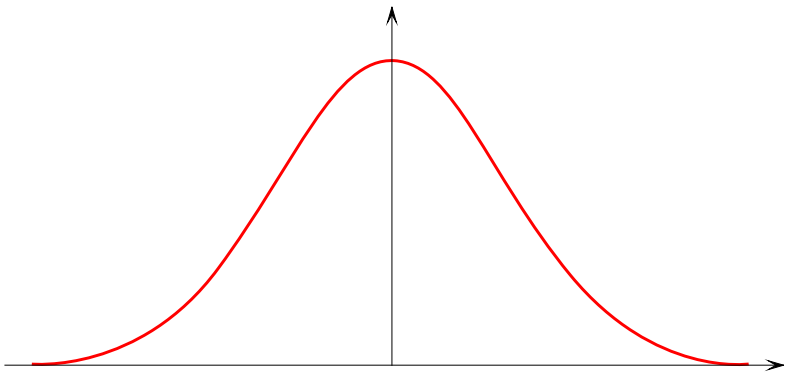


FIGURE 2.2: *The shape of the probability density function of a Gaussian distribution.*

2.1.4 The beta distribution

The *beta distribution* is continuous distribution with two interesting properties. First, it has finite support. Secondly, the shape of the distribution can be manipulated to a considerable degree of flexibility by altering its two *shape parameters* $\alpha > 0$ and $\beta > 0$. Applications of this distribution include the modelling of the duration of a task when the minimum and maximum durations are known. The probability density function for a beta distributed random variable X is given by

$$f(x) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}, \quad x \in [0, 1],$$

where $\Gamma(r) = \int_0^\infty x^{r-1}e^{-x}\,\mathrm{d}x$ is the well-known *Gamma function* with $r > 0$. A random variable Y defined over the range $[a, b]$ can be constructed by means of the linear variable transformation $Y = a + (b-1)X$ [200]. The effect of varying the shape parameters α and β on the shape of the distribution is illustrated in Figure 2.3.

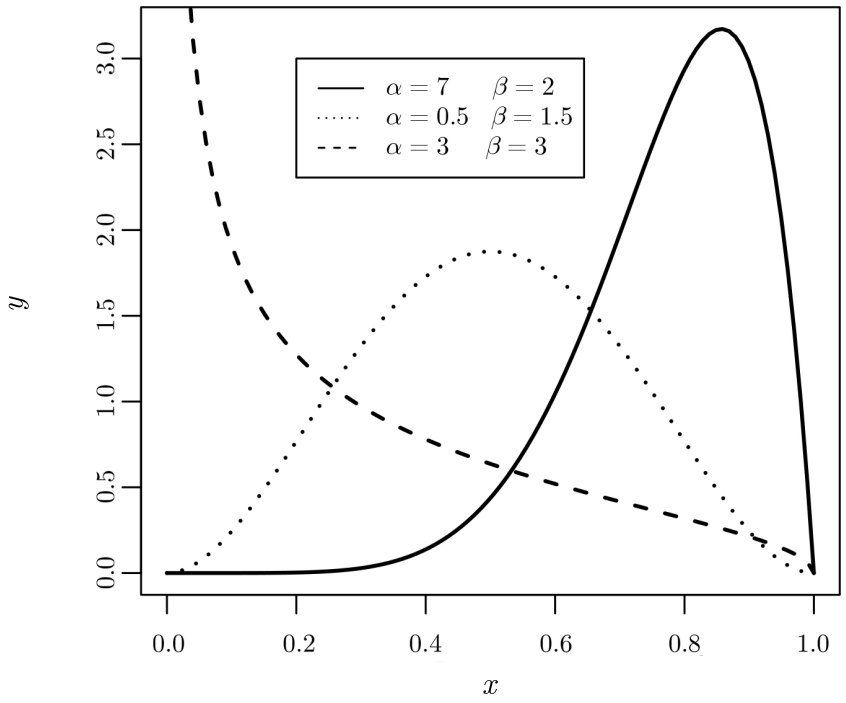


FIGURE 2.3: *The beta distribution with various combinations of shape parameter values.*

2.1.5 The Dirichlet distribution

The *Dirichlet distribution* is the multivariate generalisation of the beta distribution. Let Y^k be a vector with k elements $Y_i \geq 0$ for $i = 1, \ldots, k$ with $\sum_{i=1}^k Y_i = 1$. The Dirichlet probability density function with shape parameters $\alpha_1, \ldots, \alpha_k$ is given by

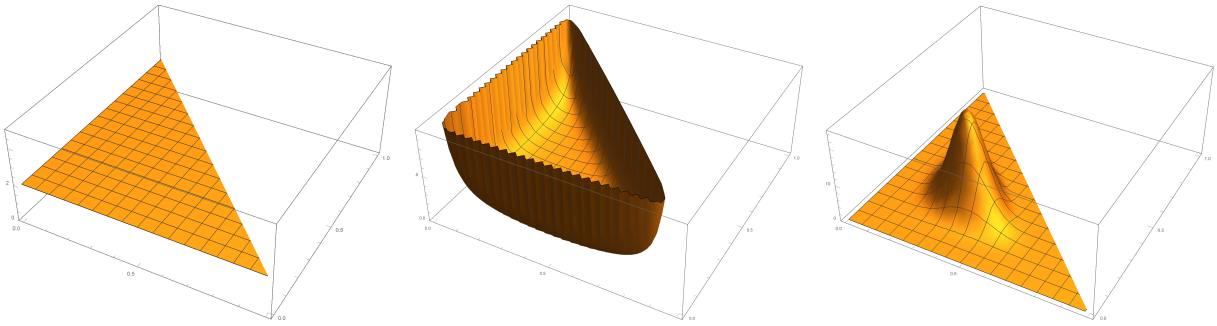
$$f(y^k) = \frac{\Gamma(\alpha_0)}{\prod_{i=1}^k \Gamma(\alpha_i)} \prod_{i=1}^k y_i^{\alpha_i - 1},$$

where $\alpha_i > 0$ for each $i \in \{1, \ldots, k\}$ and where $\alpha_0 = \sum_{i=1}^k \alpha_i$ [214].

The Dirichlet distribution is often applied in the context of *mixture modelling*. Most commonly, it is used to model the distribution of topics in a document, where each element in Y^k represents the proportional content of the document which may be assigned to topic k. The shape parameters $\alpha \in \{1, \ldots, k\}$ control how this mixture is distributed. Consider the visualisations in Figure 2.4, each of which is an example of a *symmetrical* Dirichlet distribution, where $\alpha_1 = \alpha_2 = \cdots = \alpha_k = \alpha$ (say). As is evident from the figure, the Dirichlet distribution is confined to a *simplex*[1], reflecting the bounded property of the beta distribution [30].

For $\alpha = 1$, the probability of any mixture is as likely as another, as reflected by the flat 2-simplex in Figure 2.4(a). When $\alpha < 1$, such as in the example in Figure 2.4(b), the distribution

[1] An *n-simplex* is the generalisation of a tetrahedral region in 3-space to n dimensions [337].



(a) $\alpha = 1$ (b) $\alpha = 0.1$ (c) $\alpha = 10$

FIGURE 2.4: *Visualisation of the probability density function of a symmetrical Dirichlet distribution with $k = 3$ and varying values of α. The two horizontal axes are coordinates in the plane of the simplex to which the distribution is confined, and the vertical axis corresponds to the value of the density.*

is concentrated on the outer edges of the simplex. Effectively, this means that a sample is more likely to comprise only one of the categories. Conversely, for $\alpha > 1$, the distribution forms a peak in the centre of the simplex, as is evident in Figure 2.4(c). In this instance, samples are most likely to contain a mixture of all k categories [173]. Assigning unequal values to the shape parameters results in a skewed distribution.

2.2 Basic concepts in matrix algebra

Important elementary concepts associated with matrices are introduced in this section, including the *rank* of a matrix, various matrix and vector *norms*, the notions of *eigenvalues* and *eigenvectors* associated with square matrices, as well as the process of *singular value decomposition*, which is an important construct underlying the statistical models described in the next section.

2.2.1 Matrix rank

An $n \times m$ matrix $\boldsymbol{A}$ is a rectangular array of elements arranged into n rows and m columns, in which not only the value of an element is important, but also its position in the array. An entry in the i^{th} row and j^{th} column of $\boldsymbol{A}$ is denoted by a_{ij} [42].

The *rank* of a matrix is the number of linearly independent[2] rows or columns in the matrix. If all rows and columns in a matrix are independent, it is said to be *full rank* [303]. Consider, for example, the matrices

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \end{bmatrix} \quad \text{and} \quad \boldsymbol{B} = \begin{bmatrix} 1 & 2 & 0 \\ 5 & 1 & 17 \\ 1 & 0 & 1 \end{bmatrix}.$$

The second row in $\boldsymbol{A}$ is a scalar multiple of its first row. Specifically, Row 2 = 2(Row 1). This matrix therefore has only one linearly independent row. Consequently, the rank of the matrix $\boldsymbol{A}$ is one. The same answer is returned by investigating the columns of the matrix. Since both the second and third columns are scalar multiples of the first column, the number of linearly

[2] A finite set $\mathcal{V} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k\}$ of $k \leq n$ vectors in $\mathbb{R}^n$ is *linearly dependent* if there exist real numbers $c_1, \ldots, c_k$, not all of which are zero, such that $c_1\boldsymbol{x}_1 + \ldots + c_k\boldsymbol{x}_k = \boldsymbol{0}$. The set $\mathcal{V}$ is *linearly independent* if it is not linearly dependent [342].

independent columns is also equal to one. The matrix $\boldsymbol{B}$, on the other hand, has three linearly independent rows and columns, and is therefore a full rank matrix (of rank three).

2.2.2 Norms of vectors and matrices

In order to measure the distance between vectors and matrices, the notion of a *norm* is employed. A *vector norm* on $\mathbb{R}^n$ is a function, $\|\cdot\|$, from $\mathbb{R}^n$ to $\mathbb{R}$ for which the following properties hold [42]:

(i) $\|\boldsymbol{x}\| \geq 0$ for all $\boldsymbol{x} \in \mathbb{R}^n$,

(ii) $\|\boldsymbol{x}\| = 0$ if and only if $\boldsymbol{x} = \begin{bmatrix} 0 & 0 & \dots & 0 \end{bmatrix}^T$,

(iii) $\|\alpha\boldsymbol{x}\| = |\alpha|\|\boldsymbol{x}\|$ for all $\alpha \in \mathbb{R}$ and $\boldsymbol{x} \in \mathbb{R}^n$,

(iv) $\|\boldsymbol{x} + \boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\|$ for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$.

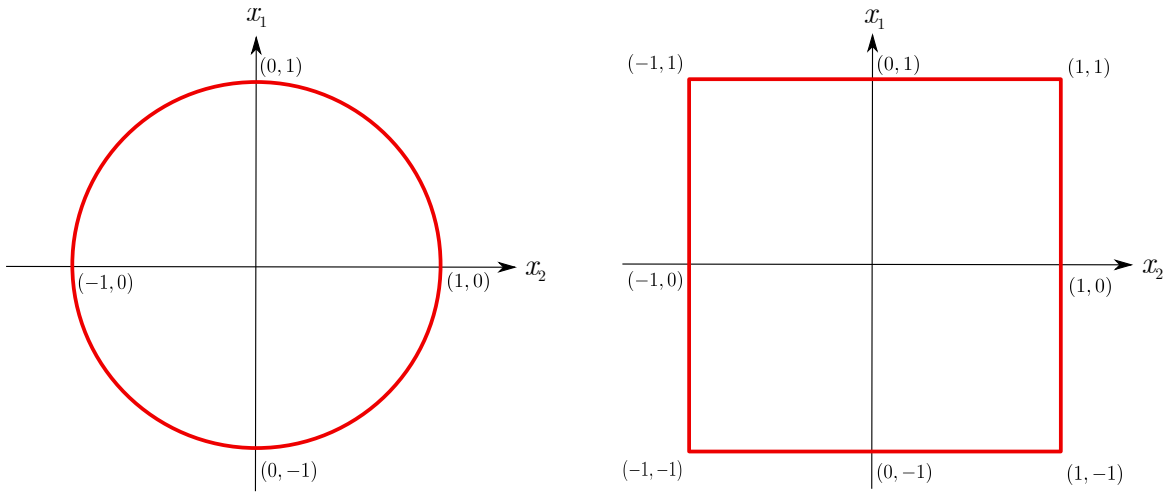
A common family of vector norms is the so-called class of *p-norms* [102], denoted by $\|\cdot\|_p$. Two of the most popular norms in this family are the ℓ_2 norm, where $p = 2$, and the ℓ_∞ norm, where $p = \infty$. The former is often also referred to as the *Euclidean norm* [42]. For a given vector, $\boldsymbol{x}$, the ℓ_2 and ℓ_∞ norms are defined as

$$\|\boldsymbol{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$$

and

$$\|\boldsymbol{x}\|_\infty = \max_{1 \leq i \leq n} |x_i|,$$

respectively. A graphical illustration of these norms in $\mathbb{R}^2$ is shown in Figure 2.5.



(a) ℓ_2 *norm* (b) ℓ_∞ *norm*

FIGURE 2.5: *Illustration of vector norms in two dimensions. Vectors lying within the red area have (a) an ℓ_2 norm or (b) an ℓ_∞ norm of at most one [42].*

Since the norm of a vector represents the distance of this vector from the origin in some sense, the distance between two vectors may be calculated as the norm of their difference. The ℓ_2 distance between the vectors $\boldsymbol{x} = \begin{bmatrix}2 & 4 & 2\end{bmatrix}^T$ and $\boldsymbol{y} = \begin{bmatrix}1 & 4 & 1\end{bmatrix}^T$, for example, may be calculated as

$$\|\boldsymbol{x} - \boldsymbol{y}\|_2 = \sqrt{(2-1)^2 + (4-4)^2 + (2-1)^2} = \sqrt{2} \approx 1.41.$$

A *matrix norm* may be defined in a similar fashion. More specifically, it is a real-valued function, $\|\cdot\|$, defined on the set of $n \times m$ matrices, for which the following properties hold [42, 102]:

(i) $\|\boldsymbol{A}\| \geq 0$,

(ii) $\|\boldsymbol{A}\| = 0$ if and only if $\boldsymbol{A}$ is the zero matrix $\boldsymbol{0}$,

(iii) $\|\alpha\boldsymbol{A}\| = |\alpha|\|\boldsymbol{A}\|$,

(iv) $\|\boldsymbol{A} + \boldsymbol{B}\| \leq \|\boldsymbol{A}\| + \|\boldsymbol{B}\|$,

(v) $\|\boldsymbol{A}\boldsymbol{B}\| \leq \|\boldsymbol{A}\|\|\boldsymbol{B}\|$,

for all $n \times m$ matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ and all real numbers α. It should be noted that the last of these properties is not always included in the definition [102]. The distance between matrices in terms of this norm is the defined by $\|\boldsymbol{A} - \boldsymbol{B}\|$.

There exist several methods to obtain matrix norms. One may, for example, determine the *natural matrix norm* associated with a vector norm. Specifically, if $\|\cdot\|$ is any vector norm, then

$$\|\boldsymbol{A}\| = \max_{\|\boldsymbol{x}\|=1} \|\boldsymbol{A}\boldsymbol{x}\| \tag{2.1}$$

is a matrix norm [42]. This relationship is illustrated in the context of a two-dimensional vector $\boldsymbol{x} = \begin{bmatrix}x_1 & x_2\end{bmatrix}^T$ in Figure 2.6. The circle in the left-hand plot indicates all vectors $\boldsymbol{x}$ for which $\|\boldsymbol{x}\|_2 = 1$, as before, and the ellipse in the right-hand plot represents these vectors resulting from a product of the matrix $\boldsymbol{A}$ with each of these vectors. The vectors indicated in red are the vectors $\boldsymbol{x}$ and $\boldsymbol{A}\boldsymbol{x}$, respectively, which feature in (2.1).

Matrix norms can also be defined without making use of associated vector norms. The matrix ℓ_2 norm, for example, is also often referred to as the *Frobenius norm* [102], and is given by

$$\|\boldsymbol{A}\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m} |a_{ij}|^2}.$$

2.2.3 Eigenvalues and eigenvectors

In Figure 2.6, it was illustrated that the multiplication of a vector $\boldsymbol{x}$ by a matrix $\boldsymbol{A}$ produces another vector $\boldsymbol{A}\boldsymbol{x}$. In this case, both the magnitude and the direction of $\boldsymbol{A}\boldsymbol{x}$ are different from those of $\boldsymbol{x}$. This is generally the case, as illustrated in Figure 2.7(a).

Now consider the case where $\boldsymbol{A}$ is an $n \times n$ matrix, and the direction of $\boldsymbol{x}$ is retained upon multiplication by $\boldsymbol{A}$, such that

$$\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x} \tag{2.2}$$

for some real or complex scalar λ. In such a case (where $\boldsymbol{x}$ is non-zero), $\boldsymbol{x}$ is referred to as an *eigenvector* of $\boldsymbol{A}$ and λ is called the *eigenvalue* associated with the eigenvector [304]. The

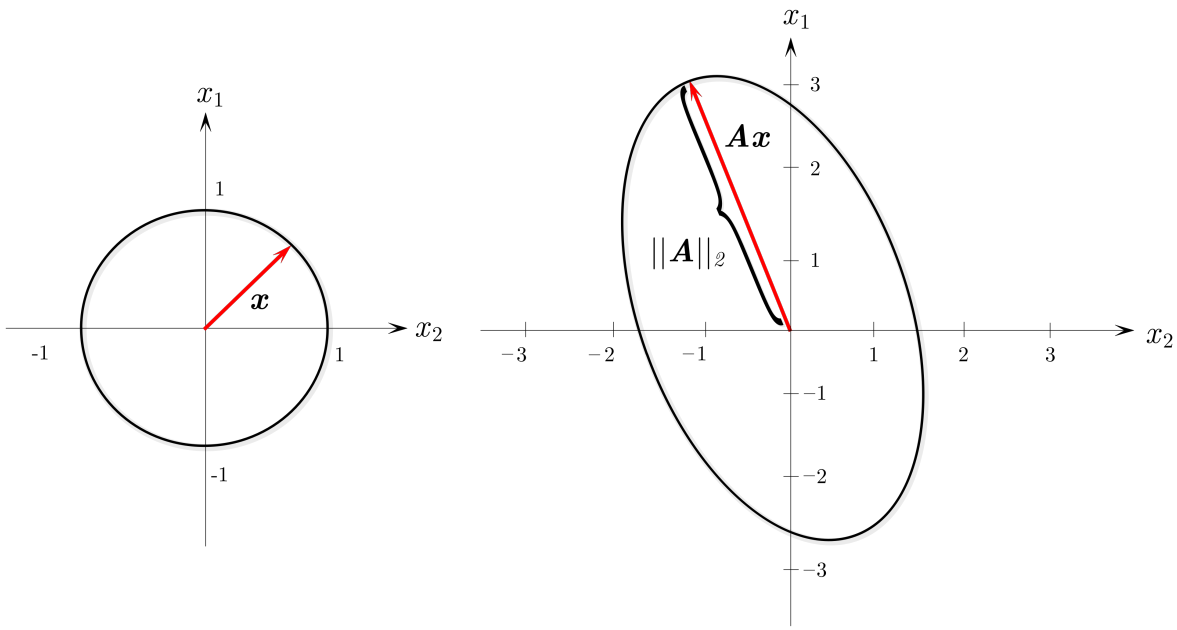


FIGURE 2.6: *Illustration of a natural ℓ_2 matrix norm in two dimensions [42]. The red vector in the left-hand plot represents these vector $\boldsymbol{x}$ for which $\|\boldsymbol{x}\|_2 = 1$ and for which the ℓ_2 norm of the vector $\boldsymbol{Ax}$ is a maximum, as indicated by the red vector in the right-hand plot. The length of this vector represents the ℓ_2 norm of the matrix $\boldsymbol{A}$.*

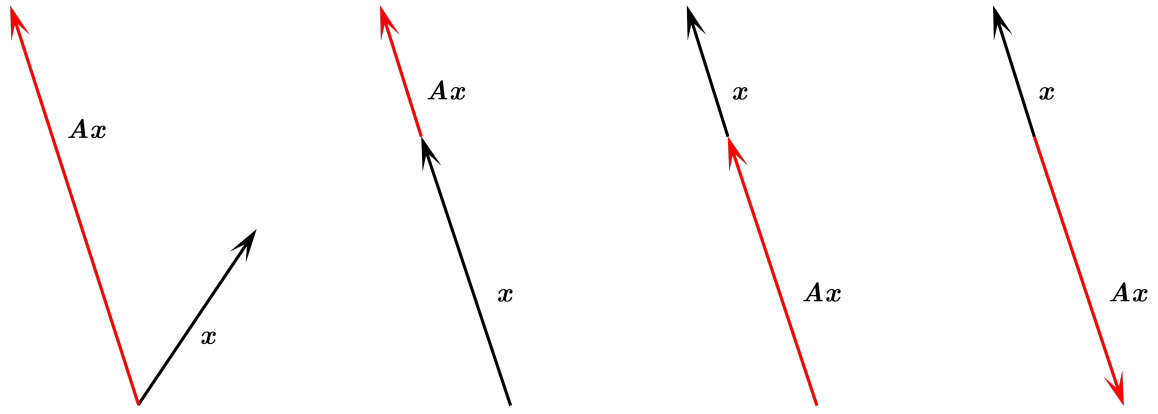


FIGURE 2.7: *The effect of an eigenvalue of a matrix $\boldsymbol{A}$. The vector $\boldsymbol{Ax}$ resulting from the multiplication of a matrix $\boldsymbol{A}$ by a vector $\boldsymbol{x}$, is shown in the cases where (a) $\boldsymbol{x}$ is not an eigenvector of $\boldsymbol{A}$, (b) $\boldsymbol{x}$ is an eigenvector of $\boldsymbol{A}$ and expansion occurs, (c) $\boldsymbol{x}$ is an eigenvector of $\boldsymbol{A}$ and contraction occurs, and (d) $\boldsymbol{x}$ is an eigenvector of $\boldsymbol{A}$ and reversal occurs (adapted from Burden [42]).*

value of a real eigenvalue λ of $\boldsymbol{A}$ determines whether the corresponding eigenvector $\boldsymbol{x}$ expands, contracts or reverses direction upon multiplication by $\boldsymbol{A}$. Expansion occurs if $\lambda > 1$, contraction occurs if $1 > \lambda > 0$, and reversal occurs if $-1 < \lambda < 0$, as is illustrated in Figure 2.7(b)–(d) [42].

Equation (2.2) may be written as $\boldsymbol{Ax} = \lambda \boldsymbol{Ix}$ or, equivalently, as

$$(\boldsymbol{A} - \lambda \boldsymbol{I})\boldsymbol{x} = \boldsymbol{0}, \tag{2.3}$$

where $\boldsymbol{I}$ is the identity matrix of the same dimension as $\boldsymbol{A}$. For each of the eigenvalues of $\boldsymbol{A}$, the determinant of the matrix $\boldsymbol{A}-\lambda\boldsymbol{I}$ becomes zero (because $\boldsymbol{x}$ is non-zero). In other words, the matrix becomes *singular* [304]. Finding the eigenvalues of $\boldsymbol{A}$ is therefore equivalent to finding the solutions of the so-called *characteristic equation* of $\boldsymbol{A}$, given by

$$|\boldsymbol{A}-\lambda\boldsymbol{I}|=0. \tag{2.4}$$

The corresponding eigenvectors may then be found by substituting each of the solutions λ_i into (2.3) and solving for the corresponding value $\boldsymbol{x}_i$ of $\boldsymbol{x}$. This procedure is illustrated in the following example, adapted from Hervé [120].

Example 2.1 *Consider the* 2×2 *matrix*

$$\boldsymbol{A}=\begin{bmatrix}2 & 3\\ 2 & 1\end{bmatrix}$$

with the characteristic equation

$$\begin{vmatrix}2-\lambda & 3\\ 2 & 1-\lambda\end{vmatrix}=(2-\lambda)(1-\lambda)-3(2)=(\lambda+1)(\lambda-4)=0. \tag{2.5}$$

Solving (2.5) yields the eigenvalues $\lambda_1=-1$ *and* $\lambda_2=4$*. The corresponding eigenvectors may then be found by substituting these results into (2.3). Substituting* $\lambda=\lambda_1$ *into (2.3) yields*

$$\begin{bmatrix}2-(-1) & 3\\ 2 & 1-(-1)\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}=\begin{bmatrix}0\\ 0\end{bmatrix} \quad\Longleftrightarrow\quad 3x_1+3x_2=0 \quad \textit{and} \quad 2x_1+2x_2=0.$$

The solution of this system of equations is the eigenvector $\boldsymbol{x}_1=\begin{bmatrix}1 & -1\end{bmatrix}^T$ *associated with* λ_1*. Similarly, the eigenvector associated with* λ_2 *may be determined as* $\boldsymbol{x}_2=\begin{bmatrix}3 & 2\end{bmatrix}^T$*.* □

If an $n\times n$ matrix $\boldsymbol{A}$ has n linearly independent eigenvectors, $\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n$, then $\boldsymbol{A}$ may be *diagonalised* as $\boldsymbol{S}^{-1}\boldsymbol{A}\boldsymbol{S}=\boldsymbol{\Lambda}$, where $\boldsymbol{S}$ is the matrix generated through the column-wise concatenation of $\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n$ and $\boldsymbol{\Lambda}$ is a diagonal matrix whose values are the eigenvalues of $\boldsymbol{A}$. Equivalently, one may write

$$\boldsymbol{A}=\boldsymbol{S}\boldsymbol{\Lambda}\boldsymbol{S}^{-1}. \tag{2.6}$$

The expression in (2.6) is known as the *eigenvalue-eigenvector factorisation* of $\boldsymbol{A}$ or, simply, the *eigen decomposition* of the matrix [120, 304]. This factorisation has various applications, and is often used to compute powers of matrices or solve differential equations more easily [304]. The eigenvalue-eigenvector factorisation of the matrix $\boldsymbol{A}$ in Example 2.1 is illustrated in the following example.

Example 2.2 *Consider again the matrix* $\boldsymbol{A}$ *of Example 2.1. Given the eigenvectors* $\boldsymbol{x}_1=\begin{bmatrix}1 & -1\end{bmatrix}^T$ *and* $\boldsymbol{x}_2=\begin{bmatrix}3 & -2\end{bmatrix}^T$ *computed in that example, the matrix* $\boldsymbol{S}$ *in the eigen decomposition of* $\boldsymbol{A}$ *is*

$$\boldsymbol{S}=\begin{bmatrix}1 & 3\\ -1 & 2\end{bmatrix}.$$

It can be verified that

$$\boldsymbol{S}^{-1}=\begin{bmatrix}2/5 & -3/5\\ 1/5 & 1/5\end{bmatrix}.$$

With $\lambda_1=-1$ *and* $\lambda_2=4$ *as the diagonal entries of* $\boldsymbol{\Sigma}$*, the diagonalisation in (2.6) becomes*

$$\boldsymbol{S}\boldsymbol{\Lambda}\boldsymbol{S}^{-1}=\begin{bmatrix}1 & 3\\ -1 & 2\end{bmatrix}\begin{bmatrix}-1 & 0\\ 0 & 4\end{bmatrix}\begin{bmatrix}2/5 & -3/5\\ 1/5 & 1/5\end{bmatrix}=\begin{bmatrix}-1 & 12\\ 1 & 8\end{bmatrix}\begin{bmatrix}2/5 & -3/5\\ 1/5 & 1/5\end{bmatrix}=\begin{bmatrix}2 & 3\\ 2 & 1\end{bmatrix}=\boldsymbol{A}. \quad \square$$

2.2.4 Singular Value Decomposition

Singular Value Decomposition (SVD) is another matrix factorisation method closely related to the eigenvalue-eigenvector factorisation [303]. Unfortunately, not all matrices for which such a factorisation is desirable fulfil the conditions set out by the eigen decomposition. The method of SVD was designed to address this problem by generalising the diagonalisation in (2.6) to any rectangular matrix [303, 304].

It can be shown that any $m \times n$ matrix $\boldsymbol{A}$ can be factored as

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T, \tag{2.7}$$

where $\boldsymbol{U}$ and $\boldsymbol{V}^T$ are orthogonal matrices[3], and $\boldsymbol{\Sigma}$ is a diagonal matrix [78, 303, 304, 322]. The expression (2.7) implies that

$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{V}\boldsymbol{\Sigma}^2\boldsymbol{V}^T \quad \text{and} \quad \boldsymbol{A}\boldsymbol{A}^T = \boldsymbol{U}\boldsymbol{\Sigma}^2\boldsymbol{U}^T.$$

Therefore, the columns of $\boldsymbol{U}$ are the eigenvectors of $\boldsymbol{A}\boldsymbol{A}^T$ and the columns of $\boldsymbol{V}$ are the eigenvectors of $\boldsymbol{A}^T\boldsymbol{A}$. Furthermore, the values on the diagonal of $\boldsymbol{\Lambda}$ are the square roots of the non-zero eigenvalues of both $\boldsymbol{A}\boldsymbol{A}^T$ and $\boldsymbol{A}^T\boldsymbol{A}$. These values are referred to as the *singular values* of $\boldsymbol{A}$ [303]. The *singular vectors* in the matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ *span*[4] the column space and the row space of the original matrix $\boldsymbol{A}$, respectively [304]. An example of SVD is provided next, adapted from Hampton [113].

Example 2.3 *Consider the* 2×3 *matrix*

$$\boldsymbol{A} = \begin{bmatrix} 3 & 2 & 2 \\ 2 & 3 & -2 \end{bmatrix}.$$

The first step in finding the SVD of this matrix is to obtain its singular values by calculating the eigenvalues of

$$\boldsymbol{A}\boldsymbol{A}^T = \begin{bmatrix} 3 & 2 & 2 \\ 2 & 3 & -2 \end{bmatrix} \begin{bmatrix} 3 & 2 \\ 2 & 3 \\ 2 & -2 \end{bmatrix} = \begin{bmatrix} 17 & 8 \\ 8 & 17 \end{bmatrix}.$$

The characteristic equation of $\boldsymbol{A}\boldsymbol{A}^T$ *is*

$$(17-\lambda)(17-\lambda) - 8(8) = \lambda^2 - 34\lambda + 225 = (\lambda - 25)(\lambda - 9) = 0. \tag{2.8}$$

Consequently, the singular values of $\boldsymbol{A}$ *are* $\sigma_1 = \sqrt{25} = 5$ *and* $\sigma_2 = \sqrt{9} = 3$. *It follows from (2.3) with* $\lambda = 25$ *that*

$$\begin{bmatrix} 17-25 & 8 \\ 8 & 17-25 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \Longleftrightarrow \quad -8u_1 + 8u_2 = 0 \quad \textit{and} \quad 8u_1 + -8u_2 = 0,$$

yielding the eigenvector $\boldsymbol{u}_1 = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$ *of* $\boldsymbol{A}\boldsymbol{A}^T$, *which normalises to* $\begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}^T$. *Substituting* $\lambda_2 = 9$ *into (2.3) similarly yields the normalised eigenvector* $\boldsymbol{u}_2 = \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix}^T$ *of*

[3]The *dot product* of two real-valued vectors $\boldsymbol{x} = \begin{bmatrix} x_1 \dots x_n \end{bmatrix}^T$ and $\boldsymbol{y} = \begin{bmatrix} y_1 \dots y_n \end{bmatrix}^T$ of length n is the scalar value $\sum_{i=1}^{n} x_i y_i$ [174]. Two vectors are *orthogonal* if their dot product is zero. An orthogonal matrix $\boldsymbol{X}$ is a square matrix whose rows and columns are orthogonal unit vectors. If $\boldsymbol{X}$ is orthogonal, then $\boldsymbol{X}^T = \boldsymbol{X}^{-1}$.

[4]A set of vectors $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\}$ is said to *span* a vector space $\mathcal{V}$ if the set of all linear combinations of the vectors, $\{k_1\boldsymbol{x}_1, \dots, k_n\boldsymbol{x}_n\}$, where $k_1, \dots, k_n$ are real numbers, is equal to $\mathcal{V}$ [364].

$\boldsymbol{AA}^T$. The same procedure may be repeated for $\boldsymbol{A}^T\boldsymbol{A}$, yielding eigenvalues $\lambda_1 = 25$, $\lambda_2 = 9$ and $\lambda_3 = 0$, and the corresponding eigenvectors

$$\boldsymbol{v}_1 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \\ 0 \end{bmatrix}, \quad \boldsymbol{v}_2 = \begin{bmatrix} 1/\sqrt{18} \\ -1/\sqrt{18} \\ 4/\sqrt{18} \end{bmatrix} \quad \textit{and} \quad \boldsymbol{v}_3 = \begin{bmatrix} 2/3 \\ -2/3 \\ -1/3 \end{bmatrix}$$

after normalisation to unit length. The singular value decomposition of $\boldsymbol{A}$ may then be written as

$$\begin{aligned} \boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T &= \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix} \begin{bmatrix} 5 & 0 & 0 \\ 0 & 3 & 0 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 1/\sqrt{18} & -1/\sqrt{18} & 4/\sqrt{18} \\ 2/3 & -2/3 & -1/3 \end{bmatrix} \\ &= \begin{bmatrix} 5/\sqrt{2} & 0 & 3/\sqrt{2} \\ 5/\sqrt{2} & 0 & -3/\sqrt{2} \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 1/\sqrt{18} & -1/\sqrt{18} & 4/\sqrt{18} \\ 2/3 & -2/3 & -1/3 \end{bmatrix} \\ &= \begin{bmatrix} 3 & 2 & 2 \\ 2 & 3 & -2 \end{bmatrix} \end{aligned}$$

□

2.3 Selected statistical data representation models

This section contains descriptions of three statistical models that feature in later chapters of the literature review. Each of these models, namely Principal Component Analysis, Latent Semantic Analysis and Latent Dirichlet Allocation, is concerned with finding lower-dimensional or compact representations of data.

2.3.1 Principal Component Analysis

Principal Component Analysis (PCA) is an unsupervised statistical learning technique aimed at finding a low-dimensional representation of a given data set that contains as much as possible of the variation in the original data [135, 243]. Consider the graph in Figure 2.8. The data in the figure consist of n observations in $p = 2$ dimensions or features and the data may be plotted in a two-dimensional graph as shown. It may be observed from the figure that most of the *variation* in the data lies along the long red line superimposed on the graph. This is known as the *first principal component* direction. Another significant portion of the variation is contained along the short red line in the same figure, called the *second principal component direction*. This direction is uncorrelated to the first principal component, since the lines are orthogonal. In PCA, linear transformations of the original components are considered with a view to choose a variable system in which the largest variance of the data comes to lie on the first axis [243]. In this way, fewer dimensions may be selected while still capturing as much as possible of the available information in the data.

Mathematically, the first principal component of a set of features $X_1, \ldots, X_p$ is the linear combination

$$Z_1 = \phi_{11}X_1 + \ldots + \phi_{p1}X_p$$

that contains the largest variance. The coefficients (also known as the *loadings*) $\phi_{11}, \ldots \phi_{p1}$ should be normalised such that $\sum_{j=1}^{p} \phi_{j1}^2 = 1$. The variance can otherwise be manipulated to be very large simply by assigning arbitrarily large values to these coefficients. To simplify mathematical operations, it is assumed that the data have been centred to have a mean value

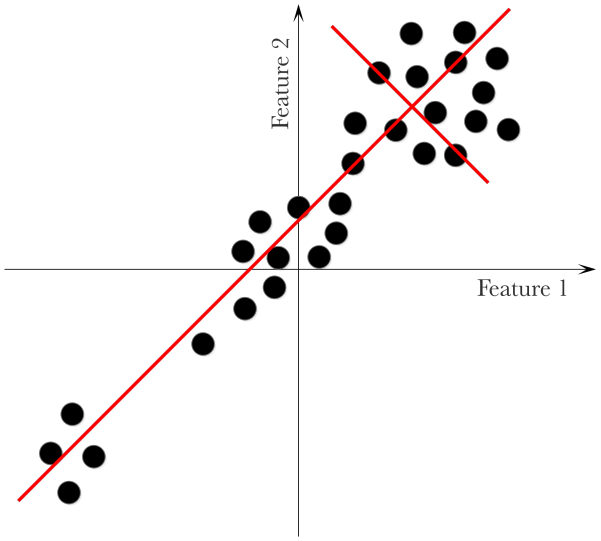


FIGURE 2.8: *A visual representation of PCA in the context of a two-dimensional data set. The long and short red lines represent the first and second principal components of the data, respectively.*

of zero. Furthermore, in order to avoid difficulties arising from variables exhibiting different *scales*, each variable is typically also scaled to have a standard deviation of one before PCA is performed [135].

The linear combination of the i-th sample in the data set is given by $z_{i1} = \phi_{11}x_{i1} + \ldots + \phi_{p1}x_{ip}$. The values $z_{11}, \ldots z_{n1}$ are referred to as the *scores* of the first principal component [135]. The general formula for variance is $\sigma^2 = \sum_{i=1}^{n}(X_i - \mu)^2/n$, where μ is the mean of the data and n is the number of data points. Given that the mean of all variables is zero, the mean value of the scores is also zero. For a data set with n observations, the variance of the scores of the k^{th} principal component is therefore $\sigma^2 = \sum_{i=1}^{n} z_{ik}^2/n$. The first principal component, Z_1, of an $n \times p$ data set $\boldsymbol{X}$ may then, finally, be calculated by solving the optimisation problem

$$\underset{\phi_{11},\ldots,\phi_{p1}}{\text{maximise}} \left\{ \frac{1}{n}\sum_{i=1}^{n}\left(\sum_{j=1}^{p}\phi_{j1}x_{ij}\right)^2 \right\} \quad \text{subject to} \quad \sum_{j=1}^{p}\phi_{j1}^2 = 1, \tag{2.9}$$

which is typically achieved using standard techniques in linear algebra, such as an eigen decomposition or SVD [135, 242].

The second principal component, Z_2, is the linear combination of $X_1, \ldots, X_p$ from the set of those linear combinations which are *uncorrelated* with Z_1, for which the variation in the scores is maximised. It can be shown that constraining Z_2 to be uncorrelated with Z_1 is equivalent to constraining the direction, or *loading vector*, $\boldsymbol{\phi}_1$, to be orthogonal to the direction $\boldsymbol{\phi}_2$ [135].

In the two-dimensional example above, there is only one possibility for $\boldsymbol{\phi}_2$. For problems in a higher dimension, however, several possibilities exist. Z_2 may then be obtained by formulating an optimisation problem similar to that in (2.9), replacing $\boldsymbol{\phi}_1$ by $\boldsymbol{\phi}_2$, and with the additional constraint that $\boldsymbol{\phi}_2$ is to be orthogonal to $\boldsymbol{\phi}_1$ [135].

2.3.2 Latent Semantic Analysis

Latent Semantic Analysis (LSA) is an unsupervised, statistical learning technique developed by Deerwester *et al.* [68] with the objective of improving *information retrieval* (IR). For this reason it is often also referred to as *Latent Semantic Indexing* (LSI). It aims to address typical issues associated with IR such as *synonymy* and *polysemy*. In the former case, users searching for information often use different words to describe the same topic (*e.g. "vehicle collision" or "car accident"*), whilst the same word may be used to refer to a different topic in the latter case (*e.g. "chip" in the context of computers or food items*) [78]. The objective of LSA is to project a *term-document matrix* onto a "semantic space," in which closely related or similar terms and documents are placed close to one another [68].

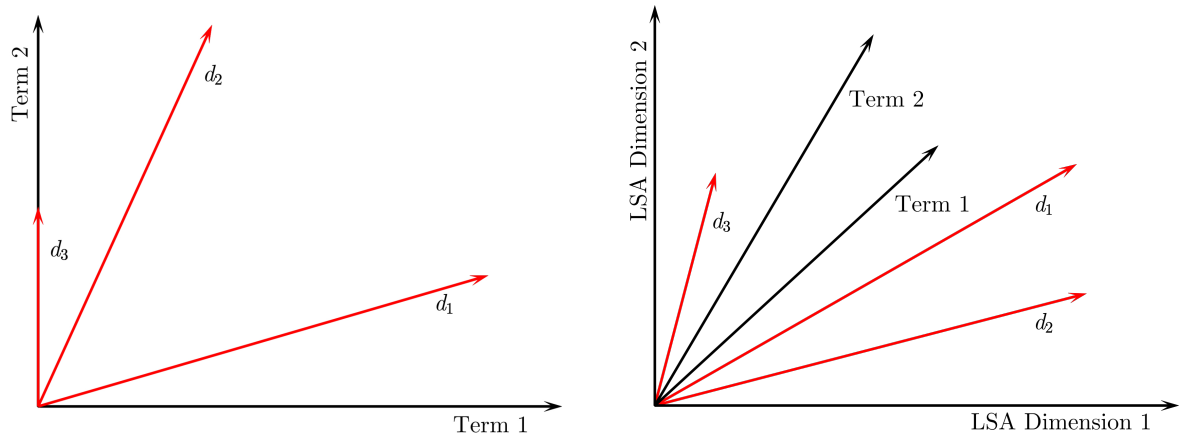
The starting point for LSA is the term-document matrix $\boldsymbol{X}$ in which each row represents a word and each column represents a document. The entries of the matrix are some frequency measure of the occurrence of each word in each document. In the next step, SVD is employed to decompose $\boldsymbol{X}$ into a product of three matrices $\boldsymbol{TSD}^T$, where the rows in $\boldsymbol{T}$ and $\boldsymbol{D}$ represent the term and document vectors in the new vector space, respectively, and the entries of the diagonal matrix $\boldsymbol{S}$ are the singular values, sorted in decreasing order [78, 322].

In order to capture only the major associative patterns that may be useful for IR, the matrix $\boldsymbol{X}$ is subsequently approximated by a reduced rank SVD, by selecting the k largest singular values in $\boldsymbol{S}$, along with their associated row vectors in $\boldsymbol{T}$ and $\boldsymbol{D}$. It can be shown that this reduced rank matrix, $\hat{\boldsymbol{X}} = \boldsymbol{T}_k\boldsymbol{S}_k\boldsymbol{D}_k^T$, is the closest approximation of $\boldsymbol{X}$ that can be achieved by a matrix of rank k, in terms of the Frobenius norm [68, 78, 322].

A geometrical interpretation of the vector representation of terms and documents is shown in Figure 2.9. In Figure 2.9(a), each document is represented as a vector in the feature space created by the terms. In Figure 2.9(b), on the other hand, both terms and documents are projected onto the linear subspace created by LSA. The similarity between terms or documents can now be calculated using this vector representation. Since the magnitude of the document vectors is often influenced by the length of a document, it is undesirable to use the magnitude of the vector difference as a measure of comparison [185]. The standard for computing the similarity between two document vectors is instead the *cosine similarity* of their vector representations [185, 322]. If the vectors are unit vectors, this value is equal to their dot product. Upon appropriate scaling of the axes, the dot product may therefore be used as a measure of similarity [68, 185].

The dot product between two rows in $\hat{\boldsymbol{X}}$ represents the extent to which two terms have a similar pattern of occurrence in the set of documents. The matrix $\hat{\boldsymbol{X}}\hat{\boldsymbol{X}}^T$ contains these term-to-term dot products. It can be shown that $\hat{\boldsymbol{X}}\hat{\boldsymbol{X}}^T = \boldsymbol{T}_k\boldsymbol{S}_k^2\boldsymbol{T}_k^T$. The similarity between two terms may therefore be calculated as the dot product between the corresponding rows in the matrix $\boldsymbol{T}_k\boldsymbol{S}_k$. Analogously, the similarity between a set of documents may be calculated as the dot product of the corresponding rows in the matrix $\boldsymbol{D}_k\boldsymbol{S}_k$ [68].

The fact that the terms are orthogonal in the representation created by the original term-document matrix causes some difficulties in practice. If d_3 is a document about cars, for example, and *Term 1* and *Term 2* are the words *"automobiles"* and *"cars,"* respectively, then the similarity between document 3 and Term 1 is zero if the document consistently uses Term 2 over Term 1, as shown in Figure 2.9(a). In the transformed vector representation in Figure 2.9(b), however, this problem is alleviated, since the term vectors are no longer orthogonal to one another [78]. In the LSA space, *Term 1* and *Term 2* are relatively close to one another, resulting in a certain degree of similarity between d_3 and *Term 1*, even though the word is not present in the document.



(a) *Original vector space representation* (b) *LSA space representation*

FIGURE 2.9: *Geometrical interpretations of (a) the term-document matrix vector space and (b) the transformed LSA vector space (adapted from Dumais [78]). The vectors d_1, d_2 and d_3 represent documents 1, 2 and 3, respectively.*

LSA and PCA are similar in that they both employ SVD to reduce the dimensionality of a large matrix. One of the major differences between these methods is the fact that the data are normalised prior to the decomposition for PCA. Geometrically, the data are therefore projected onto an *affine*[5] linear subspace for PCA [135], as opposed to the projection onto a linear subspace for LSA [33].

2.3.3 Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA) is a statistical model which, similarly to LSA, seeks to find a compact representation of the members of a *collection* that enables large collections to be processed efficiently, while retaining important information. This method, proposed by Blei *et al.* [33], is applicable to a number of problems involving collections of data made up of constituent parts. In this section, however, it is explained with reference to collections of documents made up of words, in line with the original authors' argument that this promotes an intuitive understanding of the model.

In this context, a *word* is a unit of discrete data represented as a binary vector $\boldsymbol{w}$ of length V (the *vocabulary* of available words) with entries that are all zero with the exception of the index of this word in the vocabulary. A *document* is a sequence of N words, represented by the $N \times V$ matrix $\boldsymbol{D}$, whilst a *corpus* is a collection of M documents.

LDA is a *generative* model of a corpus, which assumes that each document in the corpus exhibits a certain mixture of latent topics, and that each of these topics is characterised by a distribution over a fixed vocabulary. Furthermore, the model assumes that each document is generated according to the following process, where the number of topics k is pre-specified. The distribution of topics $\boldsymbol{\theta} = \begin{bmatrix}\theta_1 & \dots & \theta_K\end{bmatrix}$ in any given document is randomly chosen from a k-dimensional Dirichlet distribution. For each of the N words in a document, the topic z_n to which this word belongs is then randomly chosen according to the distribution of topics. The entries in $\boldsymbol{\theta}$ are the probabilities $p_1, \dots, p_k$ which define a multinomial random variable z_n. Finally, the specific word is chosen from the multinomial distribution over the vocabulary corresponding to

[5] *Affine* here means that the space need not pass through the origin [135].

the chosen topic, which is parametrised by $\boldsymbol{\beta}_{z_n} = \begin{bmatrix}\beta_1 & \dots & \beta_K\end{bmatrix}$, a row from the $k \times V$ matrix $\boldsymbol{\beta}$, which contains the distribution over the vocabulary of words for each topic [33, 56]. This generative process is summarised in Algorithm 2.1.

Algorithm 2.1: The generative process assumed by LDA

Input : The number of possible topics, k, contained in each document.
Output: A corpus.

for *each of the M documents in the corpus* **do**
  Select a distribution over topics $\boldsymbol{\theta} \sim Dir(\alpha)$;
  for *each of the N words in the document* **do**
    Select a topic z_n from $\boldsymbol{z}^k \sim Mult(k, \boldsymbol{\theta})$;
    Select a word $\boldsymbol{w}_n$ from $\boldsymbol{w}^v \sim Mult(V, \boldsymbol{\beta}_{z_n})$;
return *The generated corpus.*

The model underlying LDA may also be illustrated by means of a graphical model, as shown in Figure 2.10. The parameters α and $\boldsymbol{\beta}$ are chosen at the corpus-level, $\boldsymbol{\theta}$ is randomly drawn at the document-level for each of the M documents, and z_n and $\boldsymbol{w}$ are randomly selected for each of the N words in a document.

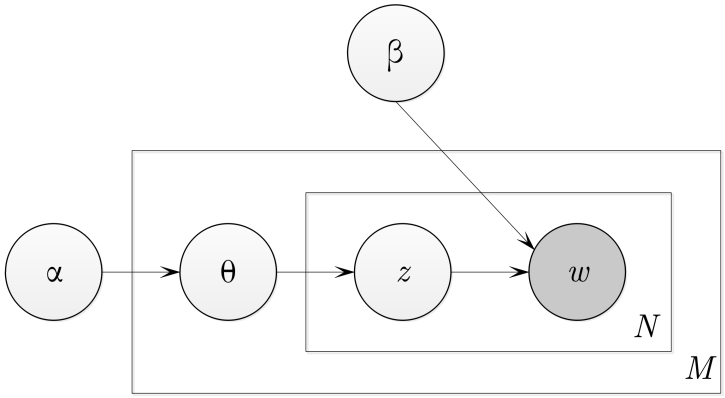


FIGURE 2.10: *Graphical model representation of LDA [33].*

The parameters α and $\boldsymbol{\beta}$ of the model are typically chosen according to the *maximum likelihood estimation* (MLE). MLE is a method for estimating the parameters of a statistical model, given some observations, by finding the parameter values that maximise the likelihood of making these observations, given the parameters [169]. In this case, the parameters are chosen such that the logarithm of the likelihood

$$\ell(\alpha, \boldsymbol{\beta}) = \sum_{i=1}^{M} \log p(\boldsymbol{D}_i | \alpha, \boldsymbol{\beta})$$

of generating each of the documents in the corpus is maximised. There is no *closed-form* solution[6] to this maximisation problem. Approximate techniques, such as the *Expectation-Maximisation algorithm*, are therefore employed to solve the problem. The fitted model can then be used to group documents according to the latent topics that they exhibit and to discover the most salient words for each topic.

[6] A solution is *closed-form* if it can be expressed in terms of functions and mathematical operations from a generally-accepted set. An infinite sum, for example, is typically not considered closed-form [302].

2.4 Solving non-linear programming problems

An *optimisation model* seeks to find values of *decision variables* that *optimise* (maximise or minimise) a given objective function among the set of all values of the decision variables that satisfy a given set of constraints [342]. A *non-linear programming problem* is an optimisation model in which the objective function is non-linear. Such problems are encountered several times in later chapters of this dissertation, where optimal parameters of statistical models are sought to minimise a given error function. In this section, methods for solving various types of these problems are described, including three gradient-based methods, the *method of Lagrange multipliers*, the *Kuhn-Tucker conditions* and *genetic algorithms*.

2.4.1 Gradient-based optimisation

In dealing with mathematical or statistical models, it is often necessary to find the minimum of a function or, in other words, to solve the optimisation problem

$$\min z = f(x_1, \ldots, x_n) \text{ subject to } (x_1, \ldots, x_n) \in \mathbb{R}.$$

The function f is *convex* if its *Hessian*[7] is *positive semi-definite*[8]. If the function f is convex, the optimal solution is the *stationary point* $\boldsymbol{x}^*$ at which

$$\frac{\partial f(\boldsymbol{x}^*)}{\partial x_1} = \ldots = \frac{\partial f(\boldsymbol{x}^*)}{\partial x_n} = 0 \quad [342].$$

If, however, the function is not convex, there is no closed form solution to the problem, and approximate methods are required to solve it. Whilst there are a multitude of valid approaches that could be employed, the focus in this section is on the *gradient-based* approach, which is typically applied when f is non-linear.

The *gradient vector* of a multivariate function $f(x_1, \ldots, x_n)$ is given by

$$\nabla f(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial f(\boldsymbol{x})}{\partial x_1} & \ldots & \frac{\partial f(\boldsymbol{x})}{\partial x_n} \end{bmatrix}.$$

The gradient vector of the function $g(x, y) = 2x + 4xy^2$, for example, is $\nabla g(x, y) = \begin{bmatrix} 2 + 4y^2 & 8xy \end{bmatrix}$. This vector defines the direction of the greatest increase of the function [342, 169]. The gradient vector of a multivariate function is analogous to the derivative of a univariate function. The *Hessian*, on the other hand, is analogous to the second derivative of a univariate function.

First-order gradient-based optimisation methods make use of the gradient vector, whilst *second-order* methods make use of the Hessian. In this section, three gradient-based optimisation algorithms are described, namely *gradient descent* (a first order method), as well as *Newton's method* and the *Broyden-Fletcher-Goldfarb-Shanno* (BFGS) algorithm (both of which are second order methods).

Gradient descent

A popular first-order method is *the method of steepest descent*, or, simply, *gradient descent* [336]. Gradient descent starts at an arbitrary point $\boldsymbol{x}_0$ and moves a certain distance ϵ away from this

[7] The *Hessian* of f is an $n \times n$ matrix containing the derivative $\partial^2 f / \partial x_i \partial x_j$ as entry in row i and column j [342].

[8] A symmetric $n \times n$ matrix is positive semi-definite if and only if its eigenvalues are non-negative [323].

point in the direction defined by the negative gradient vector at $\boldsymbol{x}_0$ in an attempt to reach the minimum point of the function. This process is repeated until a *stopping criterion* is met. Typically, the process is terminated when the norm of the gradient vector is below a given tolerance value. The general procedure is described in Algorithm 2.2 [342].

Algorithm 2.2: Generic gradient descent

Input : A starting point $\boldsymbol{x}_0$ and a stopping tolerance δ.
Output: The approximate minimum point of the function f.

1 $k \leftarrow 0$;
2 $\nabla f(\boldsymbol{x}_0) \leftarrow \left[\frac{\partial f(\boldsymbol{x}_0)}{\partial x_1} \quad \dots \quad \frac{\partial f(\boldsymbol{x}_0)}{\partial x_n}\right]$;
3 **while** $||\nabla f(\boldsymbol{x}_k)|| > \delta$ **do**
4 choose ϵ_k by minimising $f(\boldsymbol{x}_k - \epsilon_k \nabla \boldsymbol{x}_k)$ subject to $\epsilon_k \geq 0$;
5 $\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_k - \epsilon_k \nabla f(\boldsymbol{x}_k)$;
6 $k \leftarrow k + 1$;
7 $\nabla f(\boldsymbol{x}_k) \leftarrow \left[\frac{\partial f(\boldsymbol{x}_k)}{\partial x_1} \quad \dots \quad \frac{\partial f(\boldsymbol{x}_k)}{\partial x_n}\right]$;
8 **return** $\bar{\boldsymbol{x}} = \boldsymbol{x}_k$

The process of determining the optimal step size for each iteration, as shown in Line 4 of the algorithm, is referred to as a *line search*. Often, however, this step is too expensive in terms of computational complexity. In such cases, the step size is either pre-specified for each iteration, or an initial step size is given along with some method of decay. An example of such a method is the *exponential decay*, which defines the step size in each iteration as $\epsilon_k = k^{epoch}\epsilon_0$, where one *epoch* typically represents a certain number of iterations [169].

Example 2.4 *Consider the following optimisation problem*

$$\min z = f(x, y) = x^2 + 2y^2,\ x, y \in \mathbb{R}$$

and suppose the initial solution is chosen as $\boldsymbol{x}_0 = \begin{bmatrix}1 & 1\end{bmatrix}^T$ *and* $\delta = 0.1$. *The gradient vector at* $\boldsymbol{x}_0$ *is given by*

$$\nabla f(\boldsymbol{x}_0) = \begin{bmatrix}2x \\ 4y\end{bmatrix} = \begin{bmatrix}2 \\ 4\end{bmatrix}.$$

In order to find the optimal step size ϵ, *the optimisation problem*

$$\min f\left(\begin{bmatrix}1 \\ 1\end{bmatrix} - \epsilon_0 \begin{bmatrix}2 \\ 4\end{bmatrix}\right) = (1 - 2\epsilon_0)^2 + 2(1 - 4\epsilon_0)^2$$

must be solved. This is achieved by differentiating f *with respect to* ϵ_0 *and setting this derivative equal to zero. This yields the equation*

$$2(1 - 2\epsilon_0)(-2) + 4(1 - 4\epsilon_0)(-4) = 0 \tag{2.10}$$

in one unknown, which has the single solution $\epsilon_0 = 5/18$. *The update step can therefore be calculated as*

$$\boldsymbol{x}_1 = \begin{bmatrix}1 \\ 1\end{bmatrix} - \frac{5}{18}\begin{bmatrix}2 \\ 4\end{bmatrix} = \begin{bmatrix}4/9 \\ -1/9\end{bmatrix}.$$

At this point, $\nabla f(\boldsymbol{x}_1) = \begin{bmatrix}8/9 & -4/9\end{bmatrix}^T$ *with* $\|\nabla f(\boldsymbol{x}_1)\| = 0.994 > \delta$. *The optimal step length for the next iteration is again found by means of a line search as* $\epsilon_1 = 5/12$, *which yields*

$$\boldsymbol{x}_2 = \begin{bmatrix}4/9 \\ -1/9\end{bmatrix} - \frac{5}{12}\begin{bmatrix}8/9 \\ -4/9\end{bmatrix} = \begin{bmatrix}2/27 \\ 2/27\end{bmatrix}$$

with $\|\nabla f(\boldsymbol{x}_2)\| = 0.331 > \delta$. *The third iteration proceeds in the same manner and yields* $\boldsymbol{x}_3 = \begin{bmatrix}16/243 & -8/243\end{bmatrix}^T = \begin{bmatrix}0.066 & -0.033\end{bmatrix}^T$. *At this stage,* $\|f(\boldsymbol{x}_3)\| = 0.073$, *which satisfies the stopping condition. For the given tolerance, this solution is therefore 'close enough' to the optimal solution* $\boldsymbol{x}^* = \begin{bmatrix}0 & 0\end{bmatrix}^T$. □

Several variants of the gradient descent algorithm exist, most of which have been developed in the context of deep neural networks. These variants are described in the following chapter.

Newton's method

Newton's method is based on the principle that the function $f(x_1, \ldots, x_n)$ which has to be minimised may be approximated as a quadratic function, which can be minimised exactly [34, 182]. More specifically, near some point $\boldsymbol{x}_k$, f may be approximated by a second-order truncated Taylor series as

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)(\boldsymbol{x} - \boldsymbol{x}_k) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_k)^T \boldsymbol{H}(\boldsymbol{x}_k)(\boldsymbol{x} - \boldsymbol{x}_k). \tag{2.11}$$

The stationary point of (2.11) can be found by computing its partial derivatives with respect to $\boldsymbol{x}_k$ and setting it equal to zero to obtain

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{H}^{-1}(\boldsymbol{x}_k)\nabla f(\boldsymbol{x}_k). \tag{2.12}$$

Starting at some initial point, the approximate minimum point of f may be found by iteratively applying (2.12) until some stopping criterion is met, as is shown in Algorithm 2.3.

Algorithm 2.3: Newton's method

Input : A starting point $\boldsymbol{x}_0$ and a stopping tolerance δ.
Output: The approximate minimum point of the function f.

$k \leftarrow 0$;
$\nabla f(\boldsymbol{x}_0) \leftarrow \left[\frac{\partial f(\boldsymbol{x}_0)}{\partial x_1} \quad \ldots \quad \frac{\partial f(\boldsymbol{x}_0)}{\partial x_n}\right]$;
while $||\nabla f(\boldsymbol{x}_k)|| > \delta$ **do**
    Compute $\boldsymbol{H}^{-1}(\boldsymbol{x}_k)$;
    $\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_k - \boldsymbol{H}^{-1}(\boldsymbol{x}_k)\nabla f(\boldsymbol{x}_k)$;
    $k \leftarrow k + 1$;
    $\nabla f(\boldsymbol{x}_k) \leftarrow \left[\frac{\partial f(\boldsymbol{x}_k)}{\partial x_1} \quad \ldots \quad \frac{\partial f(\boldsymbol{x}_k)}{\partial x_n}\right]$;
return $\bar{\boldsymbol{x}} = \boldsymbol{x}_k$

Newton's method therefore works in the same manner as gradient descent, except that the learning rate ϵ of the latter method is replaced by the Hessian matrix. Owing to this distinction, Newton's method typically converges faster (*i.e.* in fewer iterations) than gradient descent. It can be shown, if the starting point is chosen sufficiently close to the optimum, that the order of convergence is at least two [182].

Example 2.5 *Consider again the optimisation problem in Example 2.4 with the same starting point and stopping tolerance. In addition to the gradient vector, Newton's method requires the Hessian, which is given by*

$$\boldsymbol{H} = \begin{bmatrix}\frac{\partial f^2}{\partial x \partial x} & \frac{\partial f^2}{\partial x \partial y} \\ \frac{\partial f^2}{\partial y \partial x} & \frac{\partial f^2}{\partial y \partial y}\end{bmatrix} = \begin{bmatrix}2 & 0 \\ 0 & 4\end{bmatrix}.$$

The inverse of the matrix is computed as follows

$$\boldsymbol{H}^{-1} = \frac{1}{(2)(4)-(0)(0)} \begin{bmatrix} 4 & 0 \\ 0 & 2 \end{bmatrix} = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/4 \end{bmatrix}.$$

The update step is then given by

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 - \boldsymbol{H}^{-1}\nabla f(\boldsymbol{x}_0) = \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 1/2 & 0 \\ 0 & 1/4 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Due to the fast convergence property of the method, the optimal solution to the problem was obtained in just one step in this example. □

A significant disadvantage of Newton's method is posed by its requirement to compute the inverse of the Hessian during each iteration. This calculation has an $\mathcal{O}(n^3)$ computational complexity,[9] where n is the number of variables of the function, and may be numerically unstable[10]. One manner in which this may be alleviated is to not calculate the update direction $\boldsymbol{d}_k = -\boldsymbol{H}^{-1}(\boldsymbol{x}_k)\nabla f(\boldsymbol{x}_k)$ directly, but to rather solve the linear system of equations $\boldsymbol{H}(\boldsymbol{x}_k)\boldsymbol{d}_k = -\nabla f(\boldsymbol{x}_k)$, thereby circumventing the computation of $\boldsymbol{H}^{-1}$. One popular method for solving this system of equations is the *conjugate gradient* (CG) method [121]. The update step in (2.12) is then typically expressed as

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \epsilon_k \boldsymbol{d}_k,$$

where ϵ_k is found using a line search [65]. This variation of Newton's method is referred to as the *Newton-CG* algorithm.

The Broyden-Fletcher-Goldfarb-Shanno algorithm

Although the Newton-CG variant successfully alleviates the problem of computing the inverse of the Hessian, several drawbacks of this method remain. The second-order partial derivatives needed to obtain the Hessian matrix may, for example, be difficult to compute and expensive to store. Furthermore, solving the system $\boldsymbol{H}(\boldsymbol{x}_k)\boldsymbol{d}_k = -\nabla f(\boldsymbol{x}_k)$ typically also requires $\mathcal{O}(n^3)$ operations [41].

Quasi-Newton methods seek to eliminate these difficulties by approximating the inverse of the Hessian in (2.4.1) as $\hat{\boldsymbol{H}}_k \approx \boldsymbol{H}^{-1}(\boldsymbol{x}_k)$ using gradient information. This matrix is typically initialised as the identity matrix and updated during each iteration by means of an *update matrix* $\boldsymbol{U}_k$ as

$$\hat{\boldsymbol{H}}_{k+1} = \hat{\boldsymbol{H}}_k + \boldsymbol{U}_k.$$

Ideally, this approximation should converge to the true inverse of the Hessian at the extremal point and the methods should behave similarly to Newton's method [182].

In order to approximate $\boldsymbol{H}^{-1}$ as closely as possible, $\hat{\boldsymbol{H}}$ should satisfy the following conditions [41]. First, the matrix should, as the Hessian and its inverse, be symmetric. Secondly, it should satisfy the so-called *quasi-Newton condition*

$$\boldsymbol{x}_{i+1} - \boldsymbol{x}_i = \hat{\boldsymbol{H}}_{k+1}\left(\nabla f(\boldsymbol{x}_{i+1}) - \nabla f(\boldsymbol{x}_i)\right), \text{ for } 0 \leq i \leq k.$$

[9] Let $f(n)$ be the *time complexity* of an algorithm as a function of the size n of its input data. The notation $f(n) = \mathcal{O}(g(n))$ denotes the *asymptotic upper bound* (worst case) of this time complexity [289]. For example, $\mathcal{O}(1)$ describes an algorithm whose execution time is constant regardless of the size of the input, whilst the execution time of an algorithm with an $\mathcal{O}(n)$ computational complexity increases linearly with the size of its input [22].

[10] If f is linear in some regions, for example, the Hessian can become singular (and thus non-invertible) [193].

This condition is based on the relationship

$$\boldsymbol{x} - \boldsymbol{y} = \boldsymbol{H}^{-1}\left(\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\right), \text{ for } x, y \in \mathbb{R},$$

which holds for a quadratic function of the form $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{H}\boldsymbol{x} + \boldsymbol{x}^T\boldsymbol{b} + c$, for which $\nabla f(\boldsymbol{x}) = \boldsymbol{H}\boldsymbol{x} + \boldsymbol{b}$. Lastly, $\hat{\boldsymbol{H}}_k$ should be *positive definite* (have eigenvalues that are all strictly greater than zero). If the first two conditions are satisfied, it turns out that the latter condition is automatically fulfilled [41].

One popular quasi-Newton method is the *Broyden-Fletcher-Goldfarb-Shanno* (BFGS) algorithm, which was independently discovered and published by Broyden [40], Fletcher [86], Goldfarb [101] and Shanno [279] in 1970 [233]. The update matrix is given by

$$\boldsymbol{U}_k = \left(1 + \frac{\Delta\boldsymbol{g}_k^T\hat{\boldsymbol{H}}_k\Delta\boldsymbol{g}_k}{\Delta\boldsymbol{x}_k^T\Delta\boldsymbol{g}_k}\right)\frac{\Delta\boldsymbol{x}_k\Delta\boldsymbol{x}_k^T}{\Delta\boldsymbol{x}_k^T\Delta\boldsymbol{g}_k} - \frac{\hat{\boldsymbol{H}}_k\Delta\boldsymbol{g}_k\Delta\boldsymbol{x}_k^T + (\hat{\boldsymbol{H}}_k\Delta\boldsymbol{g}_k\Delta\boldsymbol{x}_k^T)^T}{\Delta\boldsymbol{x}_k^T\Delta\boldsymbol{g}_k}, \tag{2.13}$$

where $\Delta\boldsymbol{x}_k = \nabla\boldsymbol{x}_{k+1} - \nabla\boldsymbol{x}_k$ and $\Delta\boldsymbol{g}_k = \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k)$ [41]. This computation may also be partitioned into two calculations [66], as shown in Algorithm 2.4.

Algorithm 2.4: The BFGS algorithm

Input : A starting point $\boldsymbol{x}_0$ and a stopping tolerance δ.
Output: The approximate minimum point of the function f.

1 $k \leftarrow 0$;
2 $\nabla f(\boldsymbol{x}_0) \leftarrow \left[\frac{\partial f(\boldsymbol{x}_0)}{\partial x_1} \quad \dots \quad \frac{\partial f(\boldsymbol{x}_0)}{\partial x_n}\right]$;
3 $\hat{\boldsymbol{H}}_0 \leftarrow \boldsymbol{I}$;
4 **while** $||\nabla f(\boldsymbol{x}_k)|| > \delta$ **do**
5 Choose ϵ_k by minimising $f(\boldsymbol{x}_t - \epsilon_k\hat{\boldsymbol{H}}_k\nabla f(\boldsymbol{x}_0))$ subject to $\epsilon_k \geq 0$;
6 $\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_k - \epsilon_k\hat{\boldsymbol{H}}_k\nabla f(\boldsymbol{x}_k)$;
7 $\nabla f(\boldsymbol{x}_{k+1}) \leftarrow \left[\frac{\partial f(\boldsymbol{x}_{k+1})}{\partial x_1} \quad \dots \quad \frac{\partial f(\boldsymbol{x}_{k+1})}{\partial x_n}\right]$;
8 $\Delta\boldsymbol{x}_k \leftarrow \nabla\boldsymbol{x}_{k+1} - \nabla\boldsymbol{x}_k$;
9 $\Delta\boldsymbol{g}_k \leftarrow \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k)$;
10 $\tau_k \leftarrow 1 + \frac{\Delta\boldsymbol{g}_k^T\hat{\boldsymbol{H}}_k\Delta\boldsymbol{g}_k}{\Delta\boldsymbol{x}_k^T\Delta\boldsymbol{g}_k}$;
11 $\boldsymbol{U}_k \leftarrow \frac{\tau_k\Delta\boldsymbol{x}_k\Delta\boldsymbol{x}_k^T - \hat{\boldsymbol{H}}_k\Delta\boldsymbol{g}_k\Delta\boldsymbol{x}_k^T - \left(\hat{\boldsymbol{H}}_k\Delta\boldsymbol{g}_k\Delta\boldsymbol{x}_k^T\right)^T}{\Delta\boldsymbol{x}_k^T\Delta\boldsymbol{g}_k}$;
12 $\hat{\boldsymbol{H}}_{k+1} \leftarrow \hat{\boldsymbol{H}}_k + \boldsymbol{U}_k$;
13 $k \leftarrow k + 1$;
14 **return** $\bar{\boldsymbol{x}} = \boldsymbol{x}_k$

Example 2.6 *Once again, consider the optimisation problem in Example 2.4 with the same starting point and stopping tolerance. During the first iteration of the BFGS algorithm,* $\hat{\boldsymbol{H}}_0 = \boldsymbol{I}$. *Therefore,*

$$\boldsymbol{x}_1 = \begin{bmatrix}1\\1\end{bmatrix} - \epsilon_0\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\begin{bmatrix}2\\4\end{bmatrix} = \begin{bmatrix}1\\1\end{bmatrix} - \epsilon_0\begin{bmatrix}2\\4\end{bmatrix}.$$

The line search during the first iteration is therefore the same search as in Example 2.4 and the solution $\epsilon_0 = 5/18$ *leads to the same value of* $\boldsymbol{x}_1 = \begin{bmatrix}4/9 & -1/9\end{bmatrix}^T$ *with* $\nabla f(\boldsymbol{x}_1) = \begin{bmatrix}8/9 & -4/9\end{bmatrix}^T$. *Steps 8 and 9 of Algorithm 2.4 are carried out to yield*

$$\Delta\boldsymbol{x}_0 = \begin{bmatrix}4/9\\-1/9\end{bmatrix} - \begin{bmatrix}1\\1\end{bmatrix} = \begin{bmatrix}-5/9\\-10/9\end{bmatrix}$$

and

$$\Delta \boldsymbol{g}_0 = \begin{bmatrix} 8/9 \\ -4/9 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -10/9 \\ -40/9 \end{bmatrix},$$

respectively. These values are used to compute

$$\tau_0 = 1 + \frac{\begin{bmatrix} -10/9 - 40/9 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -10/9 \\ -40/9 \end{bmatrix}}{\begin{bmatrix} -5/9 & -10/9 \end{bmatrix} \begin{bmatrix} -10/9 \\ -40/9 \end{bmatrix}} = 1 + \frac{1700/81}{50/9} = \frac{43}{9},$$

which, together with $\begin{bmatrix} -5/9 & -10/9 \end{bmatrix} \begin{bmatrix} -10/9 & -40/9 \end{bmatrix}^T = 50/9$*, yields*

$$\begin{aligned} \boldsymbol{U}_0 = \frac{9}{50} \Bigg(& \frac{43}{9} \begin{bmatrix} -5/9 \\ -10/9 \end{bmatrix} \begin{bmatrix} -5/9 & -10/9 \end{bmatrix} \\ & - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -10/9 \\ -40/9 \end{bmatrix} \begin{bmatrix} -5/9 & -10/9 \end{bmatrix} - \left(\begin{bmatrix} -10/9 \\ -40/9 \end{bmatrix} \begin{bmatrix} -5/9 & -10/9 \end{bmatrix} \right)^T \Bigg) \\ = \frac{9}{50} & \left(\frac{43}{9} \begin{bmatrix} 25/81 & 50/81 \\ 50/81 & 100/81 \end{bmatrix} - \begin{bmatrix} 50/81 & 100/81 \\ 200/81 & 400/81 \end{bmatrix} - \begin{bmatrix} 50/81 & 200/81 \\ 100/81 & 400/81 \end{bmatrix} \right) \\ = & \begin{bmatrix} 7/162 & -11/81 \\ -11/81 & -58/81 \end{bmatrix}. \end{aligned}$$

The approximated inverse of the Hessian during the next iteration is given by

$$\boldsymbol{H}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 7/162 & -11/81 \\ -11/81 & -58/81 \end{bmatrix} = \begin{bmatrix} 169/162 & -11/81 \\ -11/81 & 23/81 \end{bmatrix}.$$

The update step is formulated to yield

$$\boldsymbol{x}_2 = \begin{bmatrix} 4/9 \\ -1/9 \end{bmatrix} - \epsilon_1 \begin{bmatrix} 169/162 & -11/81 \\ -11/81 & 23/81 \end{bmatrix} \begin{bmatrix} 8/9 \\ -4/9 \end{bmatrix} = \begin{bmatrix} 4/9 \\ -1/9 \end{bmatrix} - \epsilon_1 \begin{bmatrix} 80/81 \\ -20/81 \end{bmatrix}.$$

Therefore, the one-dimensional optimisation problem that must be solved is

$$\max_{\epsilon_1} \ \left(\frac{4}{9} - \frac{80}{81}\epsilon_1 \right)^2 + 2 \left(\frac{-1}{9} + \frac{20}{81}\epsilon_1 \right)^2, \ \epsilon_1 \geq 0$$

Differentiating the objective function and setting the result equal to zero yields

$$2 \left(\frac{4}{9} - \frac{80}{81}\epsilon_1 \right) \left(-\frac{80}{81} \right) + 4 \left(\frac{-1}{9} + \frac{20}{81}\epsilon_1 \right) \left(\frac{20}{81} \right) = 0,$$

from which it follows that $0.4500 = \epsilon_1$*. Finally,*

$$\boldsymbol{x}_2 = \begin{bmatrix} 4/9 \\ -1/9 \end{bmatrix} - 0.45 \begin{bmatrix} 80/81 \\ -20/81 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

with $\|\nabla f(\boldsymbol{x}_2)\| = 0$*, which satisfies the stopping condition. As mentioned previously, this is the optimal solution to the problem.* □

2.4.2 The method of Lagrange multipliers

The methods discussed in the previous section are suitable for solving an *unconstrained* optimisation problem. The *method of Lagrange multipliers* may, however, be used to solve a constrained optimisation problem in which all constraints are equality constraints, specifically problems of the type

$$\min (\text{or} \max)\ z = f(x_1, \ldots, x_n) \tag{2.14}$$

$$\text{subject to } g_1(x_1, \ldots, x_n) = b_1, \tag{2.15}$$

$$\vdots$$

$$g_m(x_1, \ldots, x_n) = b_m. \tag{2.16}$$

Say (2.14)–(2.16) is a minimisation problem. In order to solve such a problem, a *multiplier* λ_i is associated with the i^{th} constraint and the *Lagrangian*

$$L(x_1, \ldots, x_n, \lambda_1, \ldots, \lambda_m) = f(x_1, \ldots, x_n) + \sum_{i=1}^{m} \lambda_i [b_i - g_i(x_1, \ldots, x_n)] \tag{2.17}$$

is formed. Subsequently, a point $\bar{a} = (\bar{x}_1, \ldots, \bar{x}_n, \bar{\lambda}_1, \ldots, \bar{\lambda}_m)$ is sought which minimises (2.17) [299, 342]. This entails finding the stationary point by setting the partial derivatives of the Lagrangian equal to zero. If $\bar{a}$ minimises (2.17), then

$$\frac{\partial L}{\partial \lambda_i} = b_i - g_i(x_1, \ldots, x_n) = 0 \tag{2.18}$$

at this point. This shows that the constraints in (2.15)–(2.16) are satisfied. Furthermore, let

$$a' = (x'_1, \ldots, x'_n, \lambda'_1, \ldots, \lambda'_m)$$

be any point in the feasible region of (2.15). Since a' minimises (2.17), it holds that

$$L(\bar{a}) \leq L(a'). \tag{2.19}$$

Moreover, since $\bar{a}$ and a' are both feasible in (2.14), the terms involving $\lambda_1, \ldots, \lambda_m$ in (2.17) are all zero and (2.19) becomes $f(\bar{x}_1, \ldots, \bar{x}_n) \leq f(x'_1, \ldots, x'_n)$. Therefore $(\bar{x}_1, \ldots, \bar{x}_n)$ also solves (2.14)–(2.16).

In general, it holds that any point $(\bar{x}_1, \ldots, \bar{x}_n, \bar{\lambda}_1, \ldots, \bar{\lambda}_m)$ which satisfies (2.19) will yield an optimal solution $(\bar{x}_1, \ldots, \bar{x}_n)$ to (2.14)–(2.16) if $f(x_1, \ldots, x_n)$ is a convex function and each $g_i(x_1, \ldots, x_n)$ is a linear function [342].

The following example is adapted from Winston [342].

Example 2.7 *Consider the optimisation problem*

$$\max\ z = -2x^2 - y^2 + xy + 8x + 3y$$

$$\textit{subject to } 3x + y = 10.$$

In this case, $L(x, y, \lambda) = -2x^2 - y^2 + xy + 8x + 3y + \lambda(10 - 3x - y)$. *Setting* $\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} = \frac{\partial L}{\partial \lambda}$ *yields the system of equations*

$$\frac{\partial L}{\partial x} = -4x + y + 8x - 3\lambda = 0, \tag{2.20}$$

$$\frac{\partial L}{\partial y} = 2y + x + 3 - \lambda = 0, \ \textit{and} \tag{2.21}$$

$$\frac{\partial L}{\partial \lambda} = 10 - 3x - y = 0. \tag{2.22}$$

Equation (2.20) yields $y = 3\lambda - 8 + 4x$ *and (2.21) yields* $x = \lambda - 3 + 2y$. *Therefore,* $y = 3\lambda - 8 + 4(\lambda - 3 + 2y) = 7\lambda - 20 + 8y$, *or*

$$y = \frac{20}{7} - \lambda \tag{2.23}$$

and

$$x = \lambda - 3 + 2(\frac{20}{7} - \lambda) = \frac{19}{7} - \lambda. \tag{2.24}$$

Substituting (2.23) and (2.24) into (2.22) yields $10 - 3(\frac{19}{7} - \lambda) - (\frac{20}{7} - \lambda) = 0$, *or* $\lambda = \frac{1}{4}$. *Therefore, (2.23) and (2.24) yield*

$$y^* = \frac{20}{7} - \frac{1}{4} = \frac{73}{28}$$

and

$$x^* = \frac{19}{7} - \frac{1}{4} = \frac{69}{28}.$$ □

2.4.3 The Kuhn-Tucker conditions

The method of Lagrange multipliers of the previous section may be generalised to accommodate optimisation problems of the form

$$\min \text{ (or max) } z = f(x_1, \ldots, x_n) \tag{2.25}$$

$$\text{subject to } g_1(x_1, \ldots, x_n) \leq b_1, \tag{2.26}$$

$$\vdots$$

$$g_m(x_1, \ldots, x_n) \leq b_m. \tag{2.27}$$

The following theorem provides necessary conditions for a point $\bar{x} = (\bar{x}_1, \ldots, \bar{x}_n)$ to solve (2.25)–(2.27). These are commonly referred to as the *Kuhn-Tucker* (KT) conditions [342].

Theorem 2.1 *Suppose (2.25)–(2.27) is a minimisation problem. If* $\bar{x} = (\bar{x}_1, \ldots, \bar{x}_n)$ *is an optimal solution to (2.25)–(2.27), then there exist multipliers* $\bar{\lambda}_1, \ldots, \bar{\lambda}_m$ *satisfying*

$$\frac{\partial f(\bar{x})}{\partial x_j} + \sum_{i=1}^{m} \bar{\lambda}_i \frac{\partial g_i(\bar{x})}{\partial x_j} = 0, \ j = 1, \ldots, n, \tag{2.28}$$

$$\bar{\lambda}_i[b_i - g_i(\bar{x})] = 0, \ i = 1, \ldots, m, \tag{2.29}$$

$$\bar{\lambda}_i \geq 0, \ i = 1, \ldots, m. \tag{2.30}$$

For a maximisation problem, the second term in (2.28) is subtracted from the first, rather than added. As previously stated, these conditions are *necessary* for $(\bar{x}_1, \ldots, \bar{x}_n)$ to be an optimal solution to (2.25)–(2.27). The conditions *sufficient* for $(\bar{x}_1, \ldots, \bar{x}_n)$ to be an optimal solution to (2.25)–(2.27) include, in addition to the hypotheses of Theorem 2.1, that $f(\bar{x}_1, \ldots, \bar{x}_n)$ is a concave function and that $g_1(\bar{x}_1, \ldots, \bar{x}_n), \ldots, g_m(\bar{x}_1, \ldots, \bar{x}_n)$ are convex functions [342]. Similarly, for a maximisation problem, the objective function as well as the constraints must be convex functions in order for the KT conditions to be necessary and sufficient conditions for an optimal solution.

The way in which the KT conditions can be used to solve optimisation problems is best described by means of an example. The following is again adapted from Winston [342].

Example 2.8 *Consider the non-linear programming problem*

$$\begin{aligned} \max\ z = x_1(30 - x_1) + x_2(50 - x_2) - 3x_1 - 5x_2 - 10x_3 \\ \text{subject to } x_1 + x_2 - x_3 \leq 0, \\ x_3 \leq 17.25. \end{aligned}$$

Since the objective function is a sum of convex functions (and is thus also convex), and since all the constraints are linear (and hence convex), the KT conditions may be applied to find the optimal solution to the problem. They are given by

$$30 - 2x_1 - 3 - \lambda_1 = 0, \tag{2.31}$$
$$50 - 4x_2 - 5 - \lambda_1 = 0, \tag{2.32}$$
$$10 - \lambda_1 - \lambda_2 = 0, \tag{2.33}$$
$$\lambda_1(-x_1 - x_2 + x_3) = 0, \tag{2.34}$$
$$\lambda_2(17.25 - x_3) = 0, \tag{2.35}$$
$$\lambda_1 \geq 0, \text{ and} \tag{2.36}$$
$$\lambda_2 \geq 0. \tag{2.37}$$

It is useful to note that each Lagrange multiplier λ_i must satisfy either $\lambda_i = 0$ or $\lambda_i > 0$. Therefore, the following cases may be considered when searching for the optimal values of the variables:

Case 1 $\lambda_1 = \lambda_2 = 0$. *This case cannot occur because (2.33) would be violated.*

Case 2 $\lambda_1 = 0, \lambda_2 > 0$. *If $\lambda_1 = 0$, then (2.33) implies that $\lambda_2 = -10$. This would violate (2.37).*

Case 3 $\lambda_1 > 0, \lambda_2 = 0$. *From (2.33), this would require that $\lambda_1 = 10$. Substituting this value into (2.31) and (2.32) yields $x_1 = 8.5$ and $x_2 = 8.75$, respectively. Furthermore, x_3 may be calculated as $x_1 + x_2 = 17.25$ from (2.34). The solution $x_1^* = 8.5$, $x_2^* = 8.75$, $x_3^* = 17.25$, $\lambda_1^* = 10$ and $\lambda_2^* = 0$ therefore satisfies the KT conditions.*

Case 4 $\lambda_1 > 0, \lambda_2 > 0$. *Since Case 3 yields an optimal solution, this case need not be considered.* □

2.4.4 Genetic algorithms

The working of a genetic algorithm is based on the principles of natural selection. In an implementation of such an algorithm, the decision variables of an optimisation problem are encoded in a finite-length vector (*chromosome*) whose elements (*genes*) take on values (*alleles*) from a predefined set. An initial *population* of several such chromosomes is typically generated at random. Some form of *selection*, *crossover*, *mutation* and *replacement* are then performed in order to increase the average *fitness* of the population iteratively [197, 270]. In the context of a maximisation problem, the fitness associated with a particular chromosome is the value obtained by substituting the values of the decision variables encoded by the chromosome into the objective function.

More specifically, some chromosomes are chosen from the current population for reproduction during the *selection* step. The probability of selecting a particular chromosome is typically proportional to its associated fitness value [67]. Subsequently, a crossover location is determined at random. Two *parent* chromosomes are then 'crossed over' at this location to form new

offspring chromosomes [197]. The parent chromosomes $\begin{bmatrix}1 & 1 & 1 & 1\end{bmatrix}$ and $\begin{bmatrix}0 & 0 & 0 & 0\end{bmatrix}$ may, for example, be crossed over at location 2 to yield the offspring $\begin{bmatrix}1 & 1 & 0 & 0\end{bmatrix}$ and $\begin{bmatrix}0 & 0 & 1 & 1\end{bmatrix}$. For each gene in each offspring chromosome, it is then randomly decided whether or not *mutation* should take place. If mutation does, in fact, take place, the gene's allele is altered. If each gene is a binary digit, for instance, the value of the gene is complemented (zero becomes one and one becomes zero). The probability of mutation is generally very small (*e.g.* 0.001) [197]. Finally, the fitness values of the offspring chromosomes are evaluated and chromosomes in the original population are replaced according to some predefined rule based on fitness value. The chromosomes with the lowest fitness value may, for example, be replaced by the newly generated offspring. This process is repeated until some stopping criterion is met. Examples of stopping criteria are when a maximum number of *generations* have been created or when the mean difference between the fitness of individual chromosomes in the population and the average population fitness falls below a pre-specified *convergence tolerance* [276]. The values of the decision variables encoded by the fittest chromosome ever encountered by the algorithm are then typically returned by the algorithm.

Genetic algorithms are particularly suitable for non-linear problems which have large and possibly discrete solution spaces[11] — features that increase the complexity of finding a solution by the optimisation algorithms described in previous sections [188]. Due to the probabilistic nature of the algorithm, however, optimality of the returned solution is not guaranteed.

In the remainder of this section, a simple example is employed to further illustrate the concept underlying each of the generic steps outlined above. Although many different variations of the genetic algorithm exist [197], and there is no single accepted version of the algorithm [67], most variations implement these steps in a similar manner.

Example 2.9 *Consider an optimisation problem in which the objective is to*

$$\textit{maximise } z = 2x_1 + x_2 - x_3 - x_4$$

subject to the constraint $x_1, x_2, x_3, x_4 \in \{0, 1\}$. *It is clear, by inspection, that the unique optimal solution to this problem is* $x_1^* = x_2^* = 1$ *and* $x_2^* = x_3^* = 0$, *resulting in an objective function value of* $z^* = 3$. *Nevertheless, the working of a genetic algorithm may be demonstrated effectively by means of this example. In order to apply the genetic algorithm, the decision variables of the optimisation problem may be encoded in a four-element binary vector, or chromosome, in which the entries represent the values of the decision variables* x_1, x_2, x_3 *and* x_4.

To begin, an initial population is generated randomly. An example of an initial population of size four is shown in Figure 2.11. The average population fitness, in this case, is $\bar{z} = 0.5$. *During the selection step of the algorithm, chromosomes are chosen from the initial population to serve as parent chromosomes. Here, the two fittest chromosomes (those with the largest objective function values) are selected, as denoted by the red squares in the figure.*

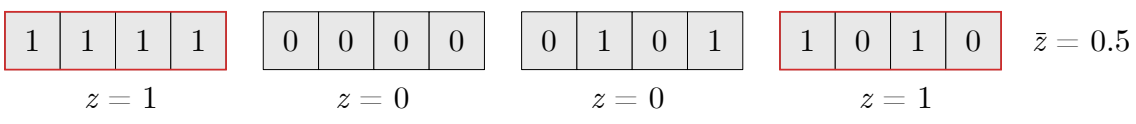


FIGURE 2.11: *A schematic representation of the selection step in a genetic algorithm. The two fittest chromosomes in the current population (highlighted in red) are selected as parents.*

[11] A *discrete* optimisation problem is one in which the set of potential solutions is finite or countably infinite [105].

During the crossover step, a crossover location is first determined randomly. Suppose the selected crossover location is 2. Each parent chromosome is then split into two sub-chromosomes — one containing all genes from position zero up to, and including, the gene at the crossover location, and one containing the remaining genes. Two offspring chromosomes are then formed by matching the first sub-chromosome of Parent chromosome 1 with the second sub-chromosome of Parent chromosome 2, and vice versa. This process is illustrated in Figure 2.12.

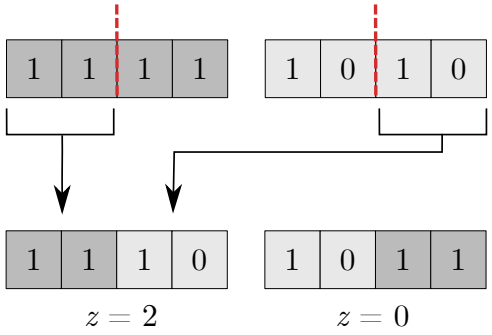


FIGURE 2.12: *A schematic representation of the crossover step in a genetic algorithm. Parent chromosomes are split at the crossover location, indicated in red, and two offspring are formed by cross-matching the split chromosomes.*

During the next step of the algorithm, the newly generated offspring are subjected to a mutation process according to some mutation probability. This may be accomplished, for example, by generating a random integer r between 0 and 100 according to a uniform distribution for each gene. The value of a gene is then altered if $r < p$, where p is the probability of mutation, expressed as a percentage. Assume that the probability of mutation in this example is 10% and that the sequences of random numbers generated for the offspring are 12, 26, 49, 88 and 91, 17, 4, 66, respectively. Only the value of the third gene in the second offspring chromosome is therefore altered. As shown in red in Figure 2.13, this is achieved in the binary case by complementing the value of the gene from one to zero.

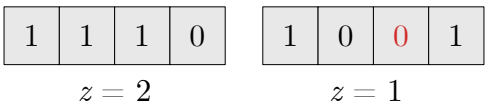


FIGURE 2.13: *A schematic representation of the mutation step in a genetic algorithm. Genes in offspring chromosomes are randomly mutated: The value of the gene highlighted in red is complemented from a one to a zero.*

Finally, certain chromosomes in the current population are replaced by the newly generated offspring to form a new generation of chromosomes. In this example, the two chromosomes with the smallest objective function values in the current population are replaced by the offspring, which are underlined in red in Figure 2.14. The new average population fitness is 1.25, a 150% improvement over the previous value of 0.5. Furthermore, the average deviation of individual chromosome fitness from this value is $(0.25+0.75+0.25+0.25)/4 = 0.375$. *Typically, this process of selection, crossover, mutation and replacement is repeated until a stopping condition is met. If the convergence tolerance in this example were set to 0.001, for instance, the algorithm would now continue to the next iteration since* $0.375 > 0.001$. *Upon satisfying the stopping condition, the values of the decision variables encoded by the fittest chromosome ever encountered by the algorithm are returned. If the algorithm were to stop at the point indicated in Figure 2.14,*

for example, the solution $x_1 = x_2 = x_3 = 1$, $x_4 = 0$ *would be returned by the algorithm with* $z = 2 < z^*$. □

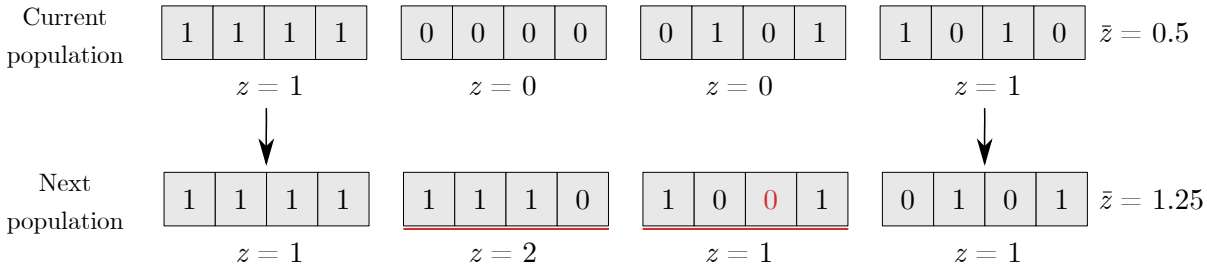


FIGURE 2.14: *A schematic representation of the replacement step in a genetic algorithm. The two least fit chromosomes in the current population are replaced by the newly generated offspring, underlined in red, to form the next generation.*

2.5 Chapter summary

This chapter opened with an introduction to random variables and statistical distributions, and this was followed by a description of the Bernoulli, binomial and multinomial (discrete) distributions, as well as the Gaussian distribution, and the beta and Dirichlet (continuous) distributions.

Subsequently, several concepts relevant to matrices were reviewed, namely the rank of a matrix, matrix norms, eigenvectors and eigenvalues, as well as the process of singular value decomposition.

The next section comprised a description of three important statistical models for the compact representation of data, namely Prinicipal Component Analysis, Latent Semantic Analysis and Latent Dirichlet Allocation.

The chapter concluded with a discussion of non-linear optimisation problems. In particular, three gradient-based optimisation algorithms for non-linear, unconstrained optimisation problems were introduced, namely gradient descent, Newton's method and the BFGS algorithm. Furthermore, algorithms for solving constrained, non-linear optimisation problems were described, namely the method of Lagrange multipliers (used to solve problems with equality constraints) and the KT conditions (used to solve problems with inequality constraints). Finally, the notion of a genetic algorithm was reviewed. This algorithm may be employed to approximate solutions to complex optimisation problems.

CHAPTER 3

Machine Learning

Contents

"I propose to consider the question, 'Can machines think?'" — Alan Turing

Machine learning is a type of artificial intelligence that provides computers with the ability to *learn* from data without being explicitly programmed [206]. More specifically, according to the widely adopted definition by Tom Mitchell [198], "[a] computer program is said to *learn* from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

A computer program that learns to *play Checkers* (T), for example, may improve its performance as measured by its *ability to win* (P) through experience acquired by *playing games against itself* (E) [198]. Or, within the realm of text classification — the domain relevant to this dissertation — one may construct a program that learns to *classify a given document* as belonging to one of a predefined set of categories (T) by analysing a *data set of texts with known categories* (E) with the aim of maximising the *proportion of correctly classified documents* (P).

In this chapter, an overview of the various types of prevailing machine learning is given, as well as an outline of the typical training procedure and evaluation metrics employed in this field. Subsequently, selected machine learning algorithms that are relevant for the understanding of the remainder of the material in this dissertation are described. The notions of *ensemble learning* and of *deep learning* algorithms are reviewed in separate sections due to their additional complexity.

3.1 Types of machine learning

Machine learning algorithms may broadly be classified as either *supervised*, *unsupervised* or *reinforcement* learning algorithms [243].

Supervised learning refers to the situation in which, for each observation $i \in \{1, \ldots, n\}$ of *predictor measurement(s)* x_i (input), there is a known *response measurement* y_i (output) [135]. These data are referred to as *labelled data* since each data point has a label corresponding to the desired output. Given a data set of natural images, for example, each image could be labelled as either *cat* or *dog*. The algorithm *learns* by predicting a label for each of a number of training observations, comparing its predictions to the *ground truth* labels and adjusting its parameters accordingly. The model can subsequently be applied to predict the labels of additional unlabelled data.

Unsupervised learning, on the other hand, is concerned with *unlabelled data* or data for which no response measurement is known. In this case, the objective of the analysis is typically to discover underlying structure within the data, such as, for example, grouping similar data points together based on their features (*clustering*) [135].

There also exist learning algorithms that are tailored to address situations in which labels are available for some, but not all, of the observations. These methods are referred to as *semi-supervised* methods and are designed to employ both labelled and unlabelled data in order to improve on the performance of purely supervised methods [60]. Intuitively, information regarding the structure or distribution of the data is extracted from all available observations in an unsupervised manner, and the relationship between these properties of the data and the class labels is investigated by means of a supervised learning component.

Weakly supervised algorithms function in a similar domain, using *"noisy"* (inexact or error-prone) labels as a basis to train a more refined model. In *image segmentation*, for example, a weak label may be represented by a *bounding box* surrounding an object of interest, such as the one in Figure 3.1(a), which can be used to train a classifier that returns a pixel-wise segmentation, as shown in Figure 3.1(b) [234].

Finally, reinforcement learning is modelled closely on the learning process of humans and other animals. Three primary components are of importance in this type of learning: The *agent* (decision maker), the *environment* (everything the agent interacts with) and *actions* (what the agent can do) [243]. The agent chooses an action in response to the observed state of its environment. It is then given feedback as to how good this decision was in the form of a *reward*, allowing it to adjust its strategy, or *policy*, accordingly. This process is repeated until an

(a) *"Noisy" bounding box segmentation* (b) *Pixel-wise segmentation*

FIGURE 3.1: *Weak supervision in image segmentation. Bounding box labels in (a) can be used as weak labels for a pixel-wise segmentation as shown in (b).*

adequate policy has been learnt. Typical applications of this technique include robotics, gaming and control systems.

Machine learning algorithms are also often classified according to the *data type* of the output variable that is to be predicted. *Regression* problems involve the prediction of a *continuous* variable, such as daily temperature, whilst *classification* problems aim to predict the value of a *discrete* variable, such as the gender of a person or the star rating of a customer review. Machine learning models that are trained to solve problems of the latter kind are therefore often referred to as *classifiers*.

Classifiers may, furthermore, be described as either *generative* or *discriminative*. The former learn a model of the *joint probability*[1] $p(x, y)$ of an observation (x, y) and use this to calculate the *posterior probability*[2] $p(y|x)$, which is used to predict the output variable y for a given input variable x. Discriminative classifiers, on the other hand, model $p(y|x)$ directly [348].

Finally, *discrete* classifiers are designed to return a distinct class (*e.g. male* or *female*), whilst *probabilistic* and *scoring* classifiers return some measure of likelihood that a sample belongs to a certain class (*e.g.* a probability score of 0.6 for *male*). Such classifiers can be transformed into discrete classifiers by applying a certain threshold, above which the sample is said to belong to a given class [84].

3.2 Model training and evaluation

As indicated previously, the source of experience for a machine learning algorithm takes the form of a data set. These data are typically split into three subsets, namely a *training data set*, a *validation data set* and a *test data set*, each of which serve a different purpose in the machine learning paradigm, as is illustrated in Figure 3.2. The training data are used to *train* the model according to the chosen machine learning algorithm in order to finding good *model parameters*.

[1]The *joint probability distribution* of two discrete random variables is a description of the set of points (x, y) along with the probability associated with each data point [200].

[2]The *posterior probability* $p(y|x)$ expresses the degree of belief of the true value of y after having observed x [200].

Most machine learning algorithms also have a set of *hyperparameters* whose values are set before training begins. These values are typically randomly initialised and then iteratively adjusted in an attempt to maximise learning performance as measured in respect of the validation data set. Lastly, the test data, also known as the *hold-out* set, is used to evaluate the final performance of the model.

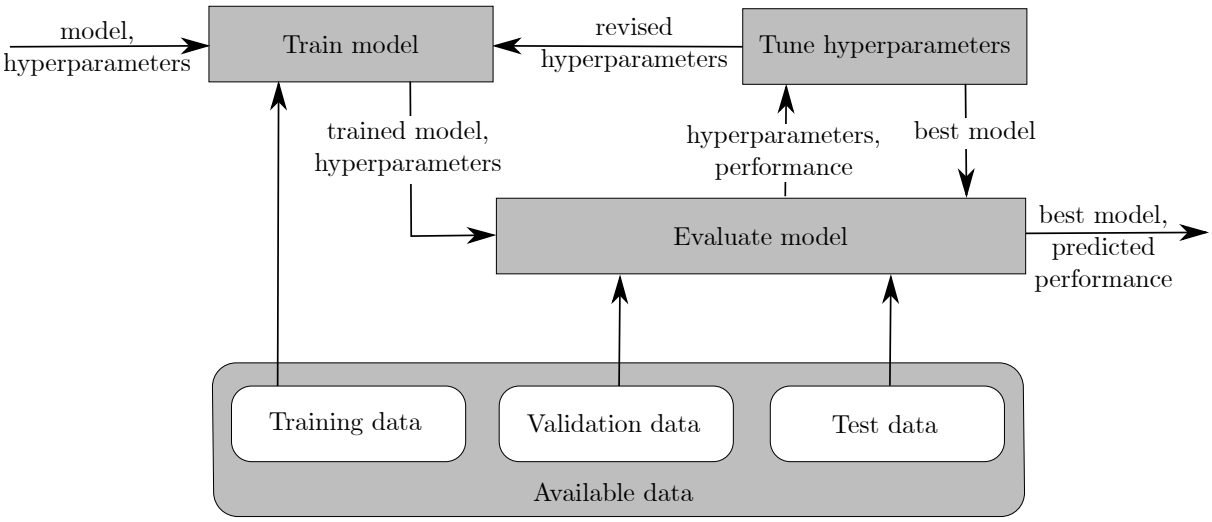


FIGURE 3.2: *The role of the training, validation and test data sets in the machine learning paradigm. The model is trained using the training data and a given set of hyperparameters. The model is then evaluated using the validation data and the result is used to adjust the hyperparameters. This process is repeated until some 'best' model is found, which is then evaluated using the test data set.*

In the remainder of this section, the process of *hyperparameter tuning* and the role of the validation and test data in estimating the generalisability of a machine learning model is explained in more detail. Subsequently, several metrics pertaining to the performance evaluation of machine learning algorithms are introduced. The focus in these sections, and indeed in the remainder of this chapter, is on the supervised learning paradigm.

3.2.1 Hyperparameter tuning

The choice of hyperparameter values can have a significant effect on the performance of a machine learning model. The process of finding good hyperparameters (*hyperparameter tuning*) is, however, nontrivial. Although more sophisticated approaches exist, including the use of optimisation methods such as *metaheuristics*[3], the most common approaches to hyperparameter tuning are *manual search*, *grid search* and *random search* [55].

In the first of these approaches, hyperparameters are chosen manually and the model is trained in respect of the training data using these hyperparameter values. The performance of the model is then evaluated in respect of the validation data set and the hyperparameters are adjusted in a way that is expected to increase model performance, based on the experience of the modeller and generally accepted heuristic guidelines. This process is repeated until a satisfactory level of

[3]A *heuristic* is a method used to solve a problem by trial and error when an exact algorithmic solution approach is impractical [342]. *Metaheuristics* are high-level strategies that guide an underlying, more problem-specific heuteristic, in an effort to increase its performance [305]. Genetic algorithms (see §2.4.4) are examples of metaheuristics.

performance is achieved. In the latter two approaches, this iterative search process is automated. In a grid search, a list of possible values is constructed for each hyperparameter. Every possible combination of values is then tested, and the one achieving the highest performance in respect of the validation data is selected. Random search works similarly, except that a *range* of values is specified for each hyperparameter rather than a distinct list of values, and random values are chosen from this range during each iteration.

3.2.2 Generalisability and the bias-variance trade-off

The purpose of the validation data set is to test the performance of a machine learning model on a portion of the data that was not seen during training, in order to gauge the model's ability to *generalise* to unseen data. If there is insufficient data to select a large enough validation data set, *k-fold cross-validation* is typically performed. To this end, the training data are partitioned into k subsets, or *folds*, and k training iterations are performed. During each iteration, the model is trained using $k-1$ folds and the remaining fold is used as a validation set. This process is illustrated in Figure 3.3. The performance of the model over all k iterations is then averaged.

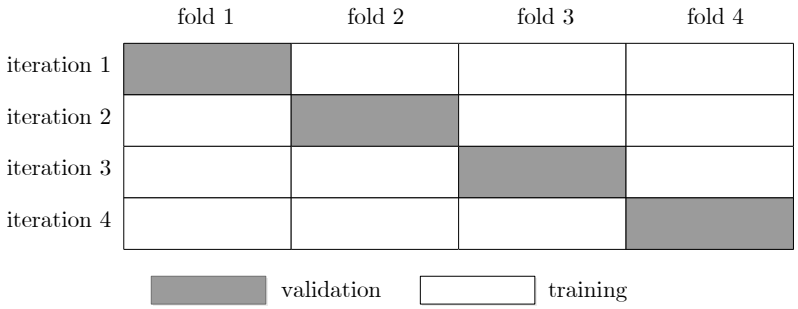
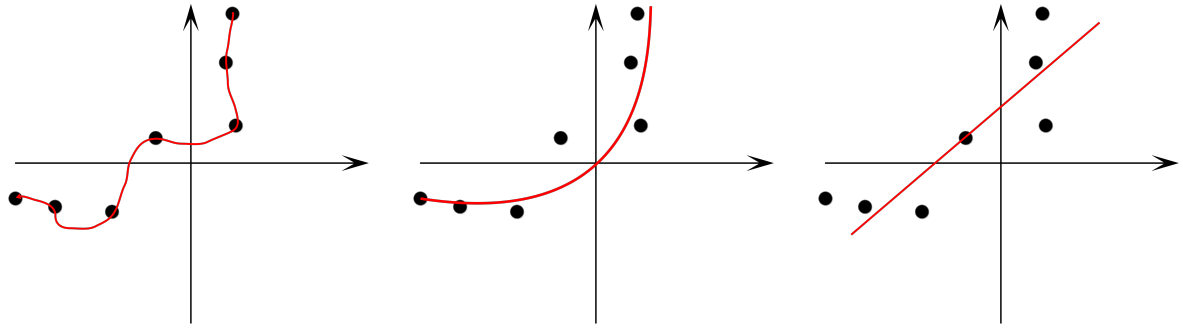


FIGURE 3.3: *A schematic illustration of 4-fold cross-validation.*

If the model performs particularly well in respect of the training data set, but poorly in respect of the validation set, it is likely *overfitting* the data. This is characterised by fitting the model to the data strictly by adding model complexity (*e.g.* using a higher-order function instead of a linear one), leading to a low bias and high variance, which attempts to account for random variation in addition to expected variation in the data. Choosing a simpler model that does not fit the training data perfectly results in a higher bias and lower variance. Such a model may perform more poorly in respect of the training data set, but may possess the ability to predict future values more accurately. If, however, the chosen model is too simple, it may *underfit* the data, and consequently be unable to explain the variance in the data adequately. This phenomenon is referred to as the *bias-variance trade-off* and is illustrated in Figure 3.4 [135].

Although the model performance in respect of the validation data set gives some indication of the model's generalisability, these data cannot be viewed as strictly unseen data, since they were used during model development (to adjust the values of hyperparameters). In order to measure model performance more accurately, it is therefore evaluated in respect of the test data set after training has been completed, in the hope that this measure resembles the model's performance upon deployment.



(a) *Overfitting* (b) *Suitable model* (c) *Underfitting*

FIGURE 3.4: *The bias-variance trade-off. Using a model that is (a) too complex or (c) too simplistic for the given data results in inferior performance.*

3.2.3 Evaluation metrics

There are a number of metrics in the literature that can be used for model evaluation. The most important evaluation metrics for classifiers are described in this section, namely *accuracy*, *precision*, *recall*, the *F-measure* and the area under a *receiver operating characteristic* (ROC) curve (AUC).

Given the predictions of a binary classifier, the purpose of which is to predict whether each observation in a set of data points belongs to the *positive* or *negative* class, and the true labels for each data point, there are four possible outcomes for each observation. If the true label of an observation is positive and it is classified as positive, it is considered a *true positive* (TP); if it is classified as negative, it is a *false negative* (FN). *True negatives* (TN) and *false positives* (FP) are defined in a similar manner. The results of the classifier may then be summarised in a so-called *confusion matrix*, as shown in Table 3.1, where each of the entries in the matrix represents the number of instances in each of the four possible categories. The entries along the diagonal represent correctly classified cases, whilst the entries off the diagonal represent errors, or cases of *confusion* [84, 135].

	Condition positive	Condition negative
Prediction positive	TP	FP
Prediction negative	FN	TN

TABLE 3.1: *A confusion matrix, showing the number of correctly classified observations on the diagonal and the number of incorrectly classified observations off the diagonal.*

This matrix may be used to calculate several common performance metrics. The *true positive rate* and *false positive rate* are defined as TP/P and FP/N, respectively, where $N = FP + TN$ and $P = TP + FN$. *Recall* is the ability to identify positive cases, defined in the same way as the true positive rate. *Precision* is the proportion of predicted positives that are, in fact, positive, or $TP/(TP + FP)$. The *F-measure* is a combined metric of precision and recall, given by

$$F = \frac{2}{\frac{1}{precision} + \frac{1}{recall}}.$$

This value is scaled between zero and one, where a value of one represents a perfect classifier. Lastly, *accuracy* is the proportion of correctly classified instances over both positive and negative

classes [84, 246], calculated as

$$\frac{TP + TN}{P + N}.$$

A measure of performance that has become popular in machine learning circles is the ROC curve, which depicts the trade-off achieved between the false positive rate, plotted on the horizontal axis, and the true positive rate on the vertical axis. Figure 3.5 is an example of such a curve.

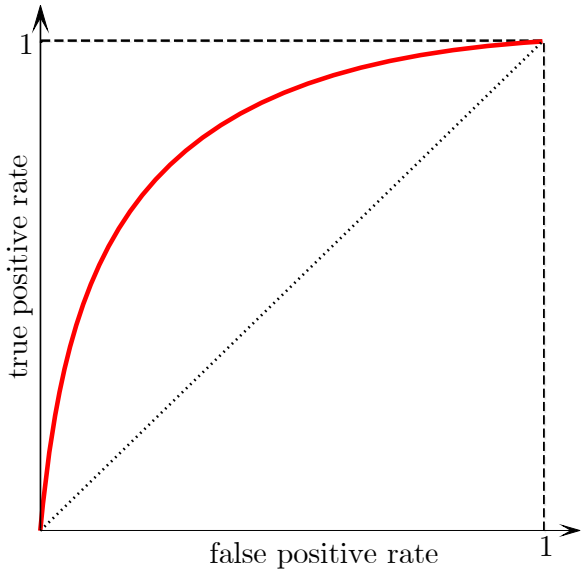


FIGURE 3.5: *The ROC curve of a classifier. The dotted line across the diagonal represents the curve equivalent to the performance of random guessing, while the dashed horizontal line represents an ideal classifier.*

The results of discrete classifiers are represented in the ROC graph as points in the plane. Classifiers in the top left-hand corner perform better (high true positive rate, low false positive rate), whilst those in the bottom right-hand corner perform poorly (low true positive rate, high false positive rate). Classifiers on the diagonal of the graph correspond to the performance of random guessing. Arbitrarily assigning a positive class to 50% of the observations, for example, would lead to the classifier correctly identifying half of the positive samples, but also incorrectly classifying half of the negative samples as positive, resulting in both a false positive and false negative rate of 0.5. Similarly, a classifier which arbitrarily assigns the positive class to 70% of the observations would be represented by the point $(0.7, 0.7)$ in the ROC space.

For scoring and probabilistic classifiers, an *ROC curve* may be constructed, where each point on the curve corresponds to a selected threshold. For instance, consider a binary classifier that predicts the scores of 0.1, 0.5 and 0.8 for three observations, which have the true class labels *negative*, *positive* and *positive*, respectively. For a threshold of zero, all three observations would be classified as positive, yielding a true positive rate of $2/2 = 1$ and a false positive rate of $1/1 = 1$, corresponding to the point $(1, 1)$ in ROC space. With a threshold of 0.6, the first two samples are classified as negative and the last sample as positive (since $0.8 > 0.6$), yielding a true positive rate of $1/2$ and a false positive rate of zero, corresponding to the point $(0, 0.5)$ on the ROC graph. This process may be repeated for several values of the threshold, thus constructing a curve. A perfect classifier, then, exhibits the unit step function as its ROC curve (separating the classes so well that either the false positive rate or the true positive rate are zero for every threshold), whilst the ROC curve of a classifier which guesses classes randomly is the identity line.

In order to compare classifiers, the area under the ROC curve is calculated, yielding an AUC value between 0 and 1. Since the diagonal represents random guessing, however, no realistic classifier should have an AUC less than 0.5 [84, 246].

The AUC metric has several advantages over other metrics, such as accuracy and the F-measure. It does not suffer from the same sensitivity to changes in the class distribution, however, since it only considers correctly and falsely classified *positive* classes. For a classifier that continuously predicts the positive class, for example, the accuracy measure would vary with the proportion of positive observations in the data set. The ROC curve would, however, remain unchanged with both a true positive and false positive rate of 1. Furthermore, it is able to compare models on a *relative* scale, measuring performance by the model's ability to separate classes. This is particularly advantageous when comparing a probabilistic classifier, for which the predictions lie between 0 and 1, with a different scoring classifier, which may be designed to produce values in $[-1, 1]$ or $[0, 100]$. Evaluating these classifiers based on accuracy achieved using the same threshold would, in the words of Fawcett [84], be "meaningless."

Different metrics may, however, be more appropriate or interesting in different settings. Medical professionals may, for example, be more interested in the recall of a classifier predicting the presence of a disease, since false negatives may have severe consequences in this case. When simply comparing models to one another, on the other hand, the ability to separate classes may be more important, which is effectively captured by the AUC. It is therefore useful to measure performance by adopting several metrics and allowing for a user-defined evaluation.

3.3 Relevant machine learning algorithms

There are many machine learning algorithms in the literature, each of which is applicable to a variety of tasks. Algorithms that are important in the context of this dissertation are described in this section, focusing particularly on classification algorithms.

3.3.1 k-Nearest Neighbours

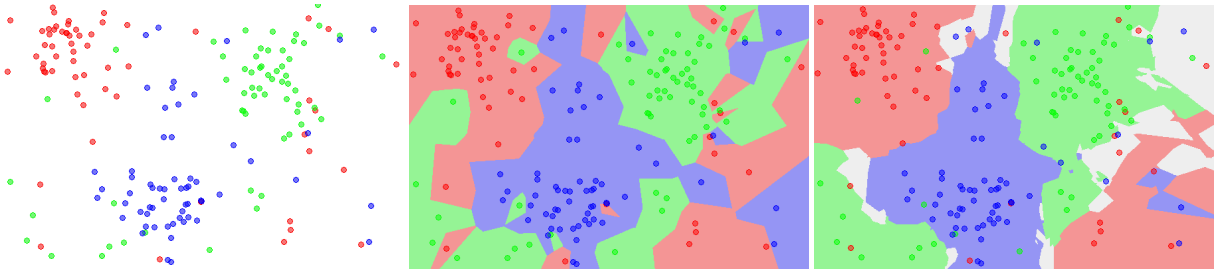
One of the simplest classification algorithms is the *k-Nearest Neighbours* (kNN) algorithm. During training, the data are sorted by some distance measure based on their features. In order to classify a new data point x_0 the k observations in the training data, which are closest to this new data point according to the adopted distance measure, are identified, represented by $\mathcal{N}_0$. The predicted class for the new observation is allocated by *voting* — the most commonly occurring class in $\mathcal{N}_0$ is allocated. Effectively, the classifier is assigning the new observation to the class for which it has the highest the conditional class probability, estimated as

$$P(Y = j|X = x_0) = \frac{1}{k} \sum_{i \in \mathcal{N}_0} I(y_i = j),$$

where $I(y_i = j)$ is an indicator variable, whose value equals 1 if $y_i = j$, or zero otherwise [135].

The kNN classifier has two hyperparameters, namely the distance measure used and the number k of neighbours considered during each iteration. The most commonly used distance measures are the ℓ_1 and ℓ_2 norms. The latter penalises large deviations more heavily than the former. The value of k controls the bias-variance trade-off in the model. Consider the data in Figure 3.6(a), with two features, plotted on the horizontal and vertical axes, respectively, and three classes, represented by the colours red, blue and green. The coloured regions in Figures 3.6(b) and

3.6(c) represent the *decision boundaries*[4] of the classes for $k = 1$ and $k = 5$, respectively. White regions represent ambiguously classified points (where votes are tied).



(a) *Data* (b) *1-NN classifier* (c) *5-NN classifier*

FIGURE 3.6: *The effect of varying the value of k in the kNN classifier. The coloured regions in (b) and (c) represent the decision boundaries for the data in (a), in the cases where $k = 1$ and $k = 5$, respectively. White regions represent ambiguous cases [297].*

For $k = 1$, the decision boundaries are complex, creating isolated regions of one class inside larger regions of another class, likely leading to incorrect predictions. For $k = 5$, on the other hand, the classifier *smooths* over these irregularities, resulting in a more generalisable model [297].

The advantages of the kNN classifier include its simplicity, especially with regard to its training procedure, and the fact that it can easily accommodate several categories. It does, however, have significant disadvantages. kNN only performs well if the data seen at the time of testing are very similar to the training data and is therefore not easily generalisable [169]. In addition, the algorithm is not well equipped to handle categorical features or high-dimensional data [243].

3.3.2 Tree-based algorithms

Tree-based methods entail segmenting the *predictor space* (the set of possible values of the input variables) into a number of distinct regions. In order to predict the output value of a new observation, it is typically assigned the *median* (most commonly occurring) or *mean* (average) value of the region to which it belongs, for instances of classification or regression problems, respectively. The rules used to partition the predictor space into these regions may be summarised in the form of a decision tree. An example of such a tree is shown in Figure 3.7.

The decision tree in the figure is based on a data set of passenger records from the infamous cruise liner, *The Titanic*, and may be used to predict whether or not a passenger would have survived the sinking of the ship, based on his or her age, gender and number of siblings present on board. According to the decision tree, any female passenger was likely to survive, whilst this fate was likely to be shared by male passengers only if they were no older than 10 and had no more than two siblings on board. Effectively, the predictor space is partitioned into four regions, namely female passengers (*Region 1*), male passengers older than 10 years (*Region 2*), male passengers 10 years or younger with more than two siblings on board (*Region 3*) and male passengers 10 years or younger with up to two siblings on board (*Region 4*) [144]. These regions are represented by the *terminal nodes* of the tree. The points along the tree where the predictor space is partitioned are *internal nodes*, the first split (*gender*, in this case) is the *root node*, and the segments connecting nodes are referred to as *branches* [181, 135].

[4]Observations which fall to one side of a *decision boundary* are allocated to a certain class, whilst observations which lie on the other side of the boundary are assigned to a different class.

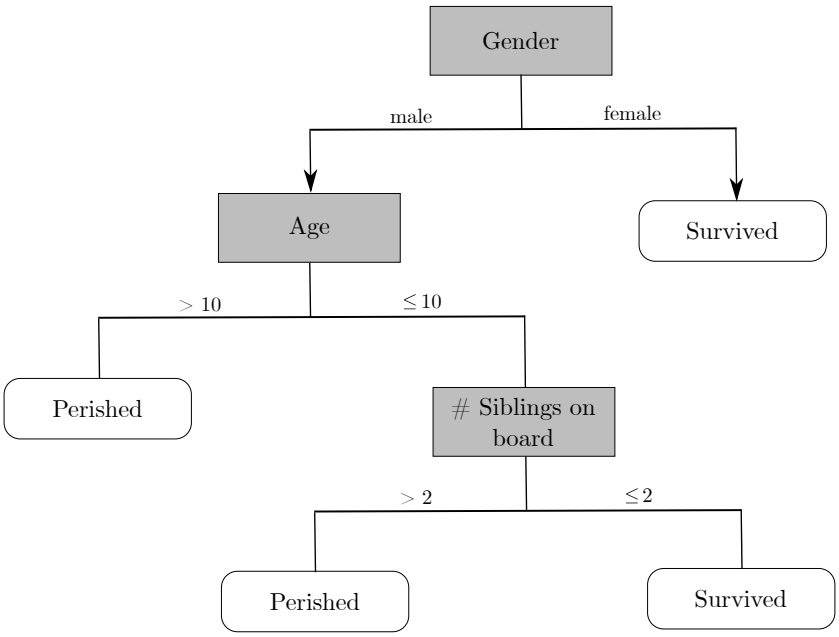


FIGURE 3.7: *A decision tree analysis by example of the passengers on-board the Titanic (adapted from Portilla [243]).*

Building and pruning decision trees

Several algorithms have been developed to construct decision trees. Two popular such algorithms are *Classification and Regression Trees* (CART) [36] and *C4.5* [251]. The approaches followed by these algorithms are similar and are therefore described together in this section, limiting the discussion to classification trees, since this is the paradigm in which the algorithms are used in this dissertation.

In the first step of the algorithms, a decision tree is generated by means of *recursive binary splitting*. Starting with the entire predictor space $X_1, \ldots, X_p$, each input variable X_j is considered along with all possible manners in which a *split* may be performed on this variable. For the *age* node in Figure 3.7, for example, each of the possible values for the passenger age would have been evaluated as a threshold α, separating the space into the regions for which $age \leq \alpha$ or $age > \alpha$. For qualitative variables, such as *gender* or *country of birth*, a binary split may be performed by, for example, assigning one category to the first branch and the remaining categories to the second branch. This procedure is employed by the CART algorithm [135]. The C4.5 algorithm, on the other hand, does not employ binary splitting for qualitative variables but, instead, creates as many branches as there are categories [158].

In order to select the best alternative, the *purity* of a node is calculated as measure of the quality of a particular split. For a classification problem with K classes, the *Gini index* is defined by

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}),$$

where $\hat{p}_{mk}$ is the proportion of training observations in the m^{th} region which originate from the k^{th} class. It can be shown that if all values of $\hat{p}_{mk}$ are close to zero or one, the Gini index takes on a small value. This suggests that observations contained in region m may be classified into one of the K classes with a high level of confidence, resulting in a high node purity. Similarly,

a small value for the *cross-entropy* measure,

$$H = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk},$$

also indicates a high-quality split [135]. In the CART algorithm, the node with the highest node purity, as measured by the Gini index, is selected. The C4.5 algorithm, on the other hand, makes use of the cross-entropy measure [181]. Using the latter measure, the optimal segmentation of the predictor space into regions maximises the *information gain*

$$Gain(S, D) = H(S) - \sum_{V \epsilon D} \frac{|V|}{|S|} H(V), \tag{3.1}$$

where S is the original predictor space, D is a partition of the space and each V is a subset of S [243]. The process of node splitting is repeated until some stopping criterion is met. Instead of partitioning the entire predictor space, however, one of the existing regions is split during each iteration. Typically, the stopping criterion takes the form of a maximum number of observations that are to be contained in each terminal node.

The decision trees resulting from this process are often excessively large and prone to overfitting [135]. Therefore, a *pruning* process is subsequently applied in order to shrink the tree to a suitably sized *subtree*. A subtree is selected such that the estimated *classification error rate*

$$E = 1 - \max_{k} \hat{p}_{mk},$$

as evaluated in respect of the validation data set or by the process of cross-validation, is minimised. Since it is impractical to evaluate each of the possible subtrees, however, alternative pruning techniques are used, including *cost complexity pruning* and *reduced error pruning*.

Reduced error pruning is a simple method, which traverses all internal nodes of the tree in a bottom-up fashion and evaluates whether replacing the node with its most frequent class results in a reduction in validation accuracy. If not, the node is replaced and the procedure is iterated until further pruning would cause a decrease in performance in respect of the validation data set [261].

In cost complexity pruning, a sequence of good subtrees is generated, yielding a smaller pool of candidates to consider as the final pruned tree. This sequence is obtained by minimising the function

$$E(T) + \alpha|T|,$$

where $E(T)$ is the error rate for a tree T and $|T|$ indicates the number of terminal nodes in T. The *tuning parameter* α controls the trade-off between the complexity of a tree and its ability to fit the training data. When $\alpha = 0$, the resulting tree is the original tree T_0 without pruning. As α increases, the size of the tree is penalised and smaller trees are more likely to minimise the function. Breiman *et al.* [36] proved that, although α could theoretically take on a continuous range of values, there is a nested and predictable sequence of pruned trees, $T_0, T_1, \ldots, T_k$, each of which is optimal for a range of α values. Consequently, only the finite number of end points of these intervals are important [36, 135]. The value of α, and its corresponding subtree, is subsequently chosen by applying cross-validation or by evaluating the performance of each tree in the sequence in respect of a validation data set [135, 261].

Bagging and Random Forests

Decision trees hold a significant advantage in that they are easily interpretable and capable of visualising the model results in a diagrammatic fashion. Furthermore, these models easily accommodate both quantitative and qualitative variables. Unfortunately, however, decision trees typically perform poorly in terms of accuracy and *robustness*[5], compared to other learning algorithms. In order to mitigate the effect of the high error rate of individual decision trees, several trees may be aggregated, using techniques such as *bagging* or *Random Forests* [135].

Bagging, or *bootstrap aggregating*, is an *ensemble learning* method first developed by Breiman [35]. The technique entails generating an *ensemble* (group or *forest*) of decision trees, each trained in respect of a different set of training data. These data sets are generated by means of *bootstrap sampling*, which involves sampling uniformly and with replacement from the original data set. A bootstrapped data set is expected to have a fraction of $1 - \frac{1}{e} (\approx 63.2\%)$ unique samples, the remainder comprising duplicate entries [35]. A prediction is made based on this ensemble by averaging the results of all decision trees (for regression) or by means of voting (for classification). Aggregating the results of several models reduces the variance in the predicted results, resulting in a superior model accuracy [135].

There is, however, one remaining concern associated with the bagging approach. If there exists one strong predictor in the data set (for which the information gain in (3.1) is particularly large) alongside several moderate predictors, many (if not all) of the generated trees will use this predictor in the root node. Consequently, the ensemble will consist of several highly correlated trees, significantly diminishing the effect of the aggregation on the model variance [135]. In order to combat this, the bagging algorithm is employed with a slight variation. Instead of training each tree on the entire feature set, a random subset of features is selected for each split, thereby generating a *random forest* of decision trees [243]. Owing to the fact that some trees are not afforded the opportunity to select the aforementioned strong predictor in the root node, the correlation between trees is successfully reduced, leading to a better predictive accuracy. For a data set with p predictors, a random sample of $\lfloor \sqrt{p} \rfloor$ features is typically selected at each node [135].

3.3.3 Support vector machines

Support vector machines are a family of supervised learning models including the *maximal margin classifier*, the *support vector classifier* and the popular *suppport vector machine* (SVM), each of which is a generalisation of the former. These models are outlined in this section, based on their descriptions by James *et al.* [135]. Subsequently, an extension of these models to non-binary classification is described.

Maximal margin classifier

In a p-dimensional space, a *hyperplane* is defined as a flat, affine subspace of dimension $p - 1$. The maximal margin classifier aims to find a hyperplane which separates data points of two different classes by as wide a margin as possible.

Consider the two-dimensional example in Figure 3.8. The data belong to one of two classes, represented by blue and pink dots, respectively. The solid black line in the figure represents a

[5]Decision trees are *non-robust* since a small change in the training data may cause a large change in the final model [135].

hyperplane which separates the two classes, while the dashed black lines delineate the *margin* by which the classes are separated. Whilst any line which lies within the margin could have been chosen as the separating hyperplane, the hyperplane depicted was chosen such that the margin, or the distance between the points closest to the boundary (the *support vectors*) in each class, is maximised.

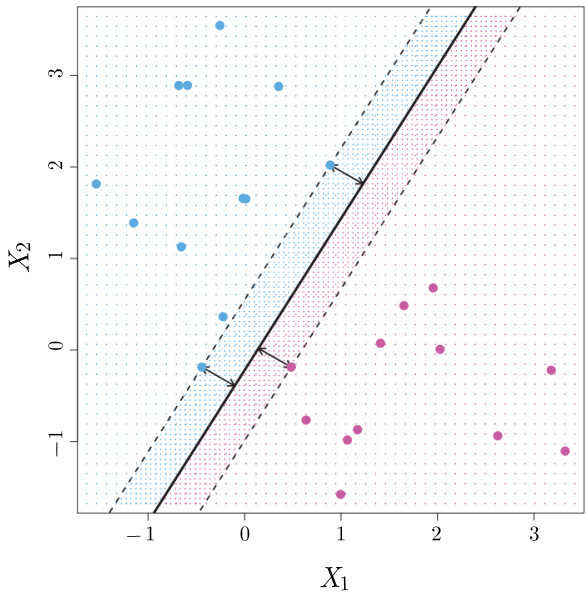


FIGURE 3.8: *The hyperplane for the maximal margin classifier in a two-dimensional example [135]. The resulting margin is indicated by means of dashed lines and the distance from the support vectors to the boundary is indicated using arrows.*

Formally, given a set of n training observations $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^p$ and associated class labels $y_1, \ldots, y_n \in \{-1, 1\}$, the maximal margin hyperplane is the solution to the optimisation problem

$$\max_{\beta_0, \beta_1, \ldots, \beta_p} M \tag{3.2}$$

$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1, \tag{3.3}$$

$$y_i(\beta_0, \beta_1 x_{i1}, \ldots, \beta_p x_{ip}) \geq M, \quad i = 1, \ldots, n, \tag{3.4}$$

where $\boldsymbol{x}_i = \begin{bmatrix} x_{i1} & \ldots & x_{ip} \end{bmatrix}$, and M represents the size of the margin of separation. A test observation, $\boldsymbol{x}^* = \begin{bmatrix} x_i^* & \ldots & x_p^* \end{bmatrix}$ is then classified based on the sign of $f(\boldsymbol{x}^*) = \beta_0 + \beta_1 x_1^* + \ldots + \beta_p x_p^*$. If it is negative, the observation is assigned to the class -1. It can be shown that, thanks to the constraint in (3.3), the perpendicular distance between the i^{th} observation and the hyperplane is given by the expression to the left of the inequality sign in (3.4). The constraint in (3.4) therefore ensures that each training observation is on the correct side of the hyperplane and at least a distance M away from the hyperplane.

Unfortunately, however, the mixture of equality and inequality constraints renders the optimisation problem difficult to solve. Several steps are necessary to simplify the problem into a solvable format. First, the parameters β_i may be expressed as a weight vector $\boldsymbol{w} = \begin{bmatrix} \beta_1 & \ldots & \beta_p \end{bmatrix}$ and a bias term $b = \beta_0$. The constraint in (3.3) is therefore equivalent to $\|\boldsymbol{w}\| = 1$. In order to eliminate this constraint, the objective function may be normalised by the norm of the weight

vector to equal $M/\|\boldsymbol{w}\|$. The expression in the objective function now represents the *geometric* (observable) margin whilst M is referred to as the *functional* margin [213].

A problem remains, however, in the non-convex nature of the objective function. It can be shown that the geometric margin is invariant to rescaling of the parameters $\boldsymbol{w}$ and b by some non-zero scaling factor, whereas the functional margin is scaled by the same factor [213, 363]. Therefore, an arbitrary scaling constraint may be imposed on the functional margin, say $M = 1$, without affecting the resulting geometric margin. This constraint can be satisfied by rescaling $\boldsymbol{w}$ and b. The objective function may therefore be changed to $1/\|\boldsymbol{w}\|$. Finally, considering the fact that maximising $1/\|\boldsymbol{w}\|$ is equivalent to minimising $\|\boldsymbol{w}\|^2$, the optimisation problem in (3.2)–(3.4) may be simplified to

$$\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2 \tag{3.5}$$

$$\text{subject to } y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1, \quad i = 1, \ldots, n. \tag{3.6}$$

This optimisation problem is a typical *quadratic programming problem* and may be solved using commercial software [213].

Support vector classifier

The data in Figure 3.8 are perfectly separable by a hyperplane. There are, however, cases where no separating hyperplane exists and the maximal margin classifier fails to find a solution to the optimisation problem in (3.5)–(3.6). In order to accommodate such cases, the maximal margin classifier may be extended to find a hyperplane that *almost* separates the classes, resulting in a *"soft margin."*

For the purpose of constructing such a classifier, the optimisation problem in (3.5)–(3.6) is modified by relaxing the constraint in (3.6) and adding a regularisation term in the objective function to yield

$$\min_{\boldsymbol{w},b,\epsilon} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\epsilon_i \tag{3.7}$$

$$\text{subject to } y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1 - \epsilon_i, \tag{3.8}$$

$$\epsilon_i \geq 0, \ i = 1, \ldots, n, \tag{3.9}$$

where C is a non-negative *tuning parameter* [135, 213]. The *slack variables* $\epsilon_1, \ldots, \epsilon_n$ allow a given observation to be on the wrong side of the margin by a certain distance, governed by C. The hyperparameter C, therefore, controls the bias-variance trade-off in the model — a large value of C results in a lower bias, but a higher variance.

Apart from being able to accommodate cases where no perfect separation can be found, this model, called the *support vector classifier*, has the advantage of being more generalisable than the maximal margin classifier, since it is not required to perfectly fit the training data. Furthermore, the support vector classifier has the interesting property that only those points which lie on the margin or violate the margin (the support vectors) affect the hyperplane. The decision rule is thus based only on a typically small subset of the training observations, making it robust to changes in observations far away from the hyperplane [135].

Support vector machine

Although the optimisation problems in (3.5)–(3.6) and (3.7)–(3.8) can be solved using quadratic programming methods, a further manipulation of these formulations can facilitate the use of *kernel functions*, which are more efficient in high-dimensional cases.

Consider the optimisation problem in (3.5)–(3.6). This problem may be converted to its *dual problem* by forming the Lagrangian

$$L(\boldsymbol{w}, b, \lambda) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{n} \lambda_i [y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) - 1], \tag{3.10}$$

as discussed in §2.4.2 [213, 363], and then maximising it with respect to the *dual variables*, the Lagrange multipliers. The Kuhn-Tucker conditions (described in §2.4.3) may be applied in order to specify the dual problem in its final form. From the first condition in (2.28), which entails computing the partial derivatives with respect to each of the *primal* variables ($\boldsymbol{w}$ and b), the equations

$$\boldsymbol{w} = \sum_{i=1}^{n} \lambda_i y_i \boldsymbol{x}_i \tag{3.11}$$

and

$$\sum_{i=1}^{n} \lambda_i y_i = 0 \tag{3.12}$$

are obtained [213, 363]. Substituting (3.11) into (3.10) yields

$$L(\lambda) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j (\boldsymbol{x}_i^T \boldsymbol{x}_j) - \sum_{i=1}^{n}\sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j (\boldsymbol{x}_i^T \boldsymbol{x}_j) - b\sum_{i=1}^{n} \lambda_i y_i + \sum_{i=1}^{n} \lambda_i,$$

which may be further simplified by utilising (3.12) to obtain

$$L(\lambda) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j (\boldsymbol{x}_i^T \boldsymbol{x}_j) + \sum_{i=1}^{n} \lambda_i.$$

The dual problem is then given by

$$\max_{\lambda} \sum_{i=1}^{n} \lambda_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j (\boldsymbol{x}_i^T \boldsymbol{x}_j) \tag{3.13}$$

$$\text{subject to} \sum_{i=1}^{n} \lambda_i y_i = 0, \tag{3.14}$$

$$\lambda_i \geq 0,\ i = 1, \ldots, n. \tag{3.15}$$

The dual problem for (3.7)–(3.8) may be derived in the same manner. It turns out that the dual formulation of this problem is the same as the dual formulation of (3.5)–(3.6), except that the constraint in (3.15) is changed to $0 \leq \lambda_i \leq C$ [213, 363]. At this stage, one can make use of the remaining Kuhn-Tucker conditions in (2.29) and (2.30), which state that if either λ_i or $y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) - 1$ is strictly positive, the other must be zero. Therefore, only data points which lie on the margin (at $f(\boldsymbol{x}) = -1$ or $f(\boldsymbol{x}) = 1$) of the decision boundary have a non-zero value of λ. Such points are called the *support vectors* and define the decision boundary. The optimal value

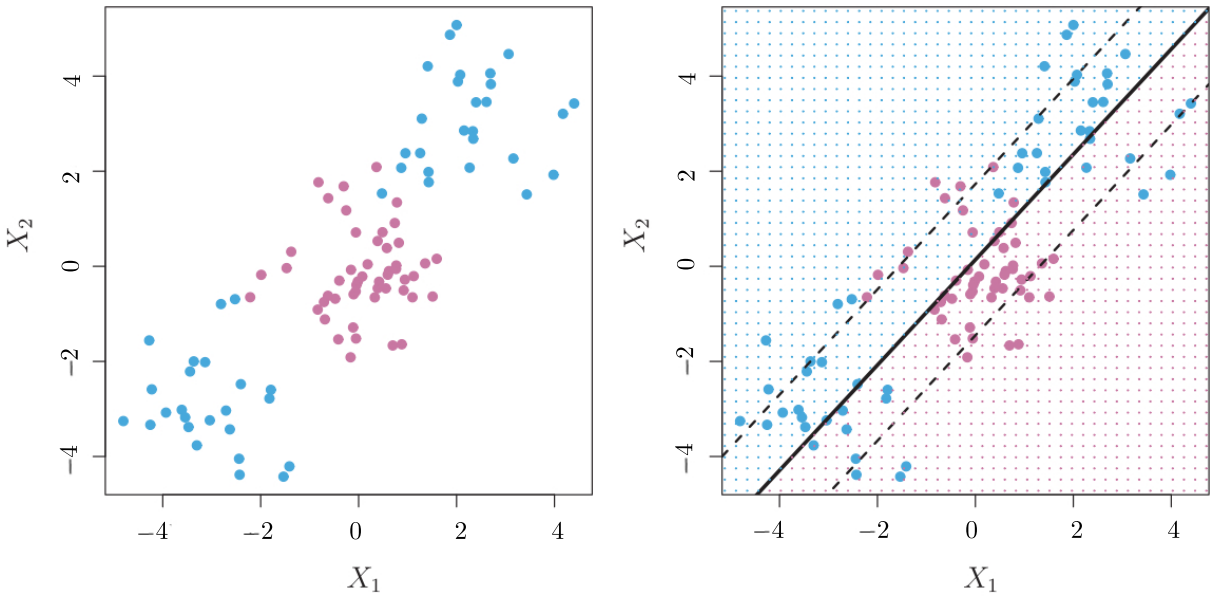
of b may be obtained using any one of the support vectors. It is, however, more numerically stable to compute the value as the mean

$$b = \frac{1}{|\mathcal{S}|} \sum_{\boldsymbol{x}_i \in \mathcal{S}} \left(y_i - \sum_{j=1}^{n} \lambda_j y_j (\boldsymbol{x}_j^T \boldsymbol{x}_i) \right),$$

where $\mathcal{S}$ denotes the set of all support vectors [363]. Furthermore, the decision rule may also be expressed only in terms of these support vectors as

$$f(\boldsymbol{x}) = \sum_{\boldsymbol{x}_i \in \mathcal{S}} \lambda_i y_i (\boldsymbol{x}^T \boldsymbol{x}_i) + b. \tag{3.16}$$

Separating data points of different classes by means of hyperplanes in this manner introduces linear boundaries between the classes. As is evident from Figure 3.9(a), however, not all data are linearly separable. The support vector classifier produces a hyperplane which acts as a poor decision boundary in such cases, as illustrated in Figure 3.9(b).



(a) *Data that are not linearly separable*

(b) *The linear boundary returned by the support vector classifier*

FIGURE 3.9: *The result returned by the support vector classifier in a case where the data are not linearly separable [135].*

In order to rectify this problem, non-linearities may be introduced into the support vector classifier by altering the decision function in (3.16) instead to read, for example,

$$y_i(\boldsymbol{w}^T \boldsymbol{x}_i + \boldsymbol{w}^T \boldsymbol{x}_i^2 + b) \geq 1 - \epsilon_i, \tag{3.17}$$

effectively enlarging the feature space from p dimensions $(X_1, \ldots, X_p)$ to $2p$ dimensions $(X_1, X_1^2, \ldots, X_p, X_p^2)$. In the new feature space, the boundary is still linear, but in the original feature space (3.17) results in a decision boundary in the form of a quadratic polynomial. One could also add higher-degree polynomials, or include interaction terms of the form $X_j X_j'$ for $j \neq j'$. Introducing non-linearities in this way would, however, result in an "unmanageabl[y]" high computational cost [135]. In contrast, the *support vector machine* extends the support vector

classifier to accommodate non-linear decision boundaries in a computationally efficient manner by employing *kernel functions*.

The *inner product* of two r-dimensional vectors $\boldsymbol{a} = \begin{bmatrix} a_1 & \dots & a_r \end{bmatrix}$ and $\boldsymbol{b} = \begin{bmatrix} b_1 & \dots & b_r \end{bmatrix}$ is defined as $\langle \boldsymbol{a}, \boldsymbol{b} \rangle = \sum_{i=1}^{r} a_i b_i$. The decision rule of the linear support vector classifier in (3.16) can therefore also be represented using only the *inner products* of the observations, as

$$f(\boldsymbol{x}) = \sum_{\boldsymbol{x}_i \in \mathcal{S}} \lambda_i y_i \langle \boldsymbol{x}^T \boldsymbol{x}_i \rangle + b. \tag{3.18}$$

The expression of the inner product in (3.18) may be replaced by the generalisation $K(\boldsymbol{x}_i, \boldsymbol{x}_i')$, where K is some function referred to as a *kernel*. The purpose of a kernel function is to quantify the similarity between two observations. Choosing the inner product as the kernel function results in the support vector classifier, as shown above. Since this results in a linear decision boundary, the inner product is known as a *linear kernel*. One could, however, choose a number of different functions to fulfil the role of kernel. A *polynomial kernel* of degree d, for example, is given by

$$K(\boldsymbol{x}_i, \boldsymbol{x}_i') = \left(a + \sum_{j=1}^{p} x_{ij} x_{ij}' \right)^d,$$

where $\boldsymbol{x}_i = \begin{bmatrix} x_{i1} & \dots & x_{ip} \end{bmatrix}$, $\boldsymbol{x}_i' = \begin{bmatrix} x_{i1}' & \dots & x_{ip}' \end{bmatrix}$ and d is a positive integer. Choosing $d > 1$ leads to a more flexible decision boundary, as shown in Figure 3.10(a). Another popular kernel function is the *radial kernel*. It takes the form

$$K(\boldsymbol{x}_i, \boldsymbol{x}_i') = \exp\left(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{ij}')^2 \right),$$

where γ is a positive constant. If a test observation is far away from a training observation, the argument of the exponential function above grows large in magnitude, and the kernel function evaluates to a very small number. Consequently, only observations close to the test observations play a role in its classification. This effect is illustrated in Figure 3.10(b).

When a support vector classifier is used in conjunction with a non-linear kernel, it is known as a support vector machine, given by [48]

$$f(\boldsymbol{x}) = \sum_{i \in \mathcal{S}} \lambda_i y_i K(\boldsymbol{x}, \boldsymbol{x}_i) + b.$$

The advantage of using kernel functions instead of working in an enlarged feature space is, on the one hand, computational in nature. The enlarged feature space is often so extensive that computations are too difficult and expensive to carry out in an efficient manner. Kernel functions in a higher-dimensional feature space, on the other hand, can also be computed without explicitly working in this feature space. Furthermore, some kernel functions, such as the radial kernel, enable calculations in an implicit feature space of infinite dimensions [135].

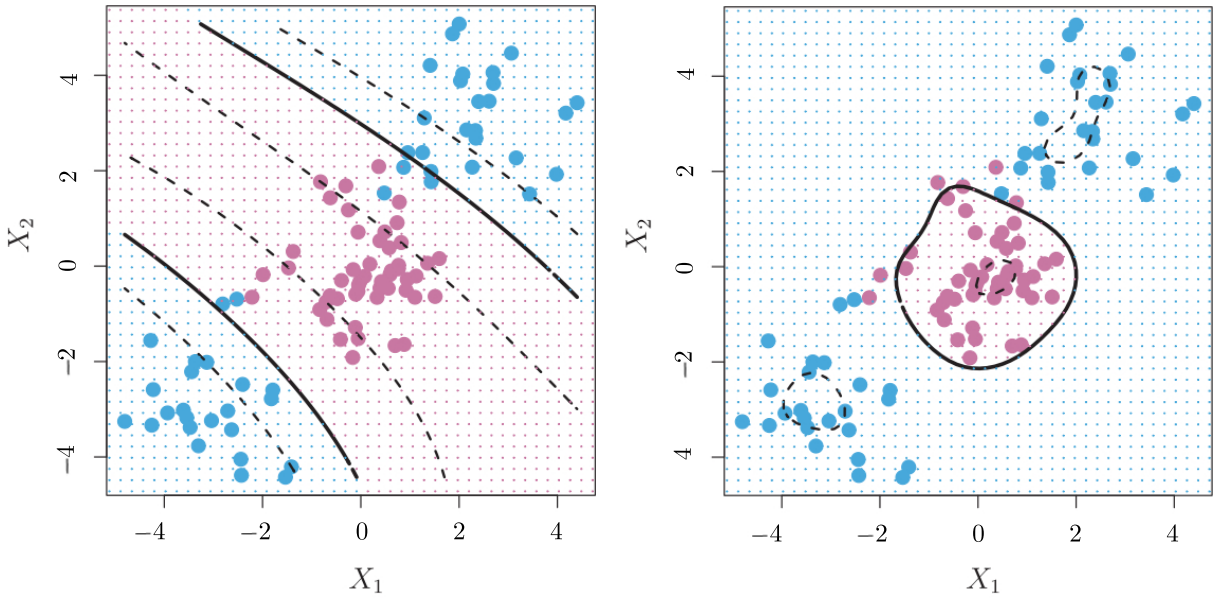
Solving for the model parameters

The dual optimisation problem associated with an SVM may be written in matrix notation as

$$\max_{\boldsymbol{\lambda}} \ \mathbf{1}^T \boldsymbol{\lambda} - \frac{1}{2} \boldsymbol{\lambda}^T \boldsymbol{Q} \boldsymbol{\lambda} \tag{3.19}$$

$$\text{subject to } \boldsymbol{y}^T \boldsymbol{\lambda} = 0, \tag{3.20}$$

$$0 \leq \lambda_i \leq C, \ i = 1, \dots, n, \tag{3.21}$$



(a) *A polynomial kernel* (b) *A radial kernel*

FIGURE 3.10: *Non-linear decision boundaries for the data in Figure 3.9(a) [135].*

where $\boldsymbol{Q}_{ij} = y_i y_j K(\boldsymbol{x}_i \boldsymbol{x}_j)$ and $\mathbf{1}$ is a vector of length n with all entries equal to one. The matrix $\boldsymbol{Q}$ is usually fully dense and often too large to be stored [48, 83]. In order to address these difficulties, the optimisation problem is typically solved using *decomposition methods*, which modify only a subset of the dual decision variables $\boldsymbol{\lambda}$ per iteration [83]. *Sequential minimal optimisation* (SMO) [239] is one popular such method, which restricts this subset, denoted by the *working set* $\mathcal{B}$, to comprise only two elements. The algorithm is given in Algorithm 3.1 [83, 213, 239].

Algorithm 3.1: Sequential minimal optimisation

Input : The dual optimisation problem (3.19)–(3.21).
Output: The optimal dual parameter values, $\boldsymbol{\lambda}^*$.

1 Find $\boldsymbol{\lambda}^1$ as an initial feasible solution
2 $k \leftarrow 1$
3 **while** $\boldsymbol{\lambda}^k$ *is not a stationary point of (3.19)–(3.21)* **do**
4 Select a two-element working set $\mathcal{B} = \{i, j\} \subset \{1, \ldots, n\}$;
5 Define $\mathcal{N} \equiv \{1, \ldots, n\} \setminus \mathcal{B}$ and $\boldsymbol{\lambda}^k_{\mathcal{B}}$ and $\boldsymbol{\lambda}^k_{\mathcal{N}}$ to be sub-vectors of $\boldsymbol{\lambda}^k$ corresponding to $\mathcal{B}$ and $\mathcal{N}$, respectively;
6 Optimise (3.19)–(3.21) with respect to $\boldsymbol{\lambda}^k_{\mathcal{B}}$ while holding $\boldsymbol{\lambda}^k_{\mathcal{N}}$ constant to find $\boldsymbol{\lambda}^{k*}_{\mathcal{B}}$;
7 $\boldsymbol{\lambda}^{k+1}_{\mathcal{B}} \leftarrow \boldsymbol{\lambda}^{k*}_{\mathcal{B}}$; $\boldsymbol{\lambda}^{k+1}_{\mathcal{N}} \leftarrow \boldsymbol{\lambda}^k_{\mathcal{N}}$;
8 $k \leftarrow k + 1$;
9 **return** $\boldsymbol{\lambda}^* \leftarrow \boldsymbol{\lambda}^k$

Since only two variables are modified in any given iteration, the optimisation problem in Step 6 of the algorithm can be computed efficiently [213]. This, however, also leads to slow convergence for difficult problems [83]. Heuristic methods are, therefore, often employed in Step 4 of the algorithm in order to accelerate the rate of convergence. The popular machine learning library

LIBSVM [48] employs a method proposed by Fan *et al.* [83], which makes use of second-order information in order to select the working set.

Non-binary classification

The intended use of support vector machines is binary classification. These algorithms may, however, be extended to accommodate several classes. The so-called *one-versus-one* approach is one manner in which this extension may be achieved.

Suppose an observation is to be classified into $K > 2$ classes. The one-versus-one approach entails constructing $\binom{K}{2}$ SVM models, each of which compares a pair of classes. The observation is then classified into one of two classes by each of these binary classifiers. The final prediction is the class to which the observation was most commonly assigned during the pairwise classification process [135, 144].

3.3.4 The naïve Bayes classifier

Naïve Bayes is a generative, probabilistic model that is often used for document classification problems [236, 316]. It is based on *Bayes' Theorem*, which states that

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)},$$

where $P(A|B)$ is the *conditional probability* of an event A, given that an event B occurs (the *posterior probability*), $P(B|A)$ is the conditional probability of B given A (the *likelihood*) and $P(A)$ and $P(B)$ are the *prior probabilities* of the events A and B, respectively [207, 282, 316].

For a given classification problem, one may want to estimate the probability that an observation belongs to a certain class y, given that it is represented by a feature vector $\boldsymbol{x} = \begin{bmatrix} x_1 & \dots & x_p \end{bmatrix}$. Using Bayes' Theorem, this can be expressed as the posterior probability

$$P(y \mid x_1, \dots, x_p) = \frac{P(y)P(x_1, \dots, x_p \mid y)}{P(x_1, \dots, x_p)}. \tag{3.22}$$

It may be difficult to consider all of the dependencies between features in order to estimate $P(x_1, \dots, x_p \mid y)$. If, however, it is assumed that each of the features $x_1, \dots, x_p$ is independent of the other features, such that

$$P(x_i \mid y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_p) = P(x_i \mid y)$$

for all i, then the relationship in (3.22) is simplified to

$$P(y \mid x_1, \dots, x_p) = \frac{P(y)\prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \dots, x_p)} \quad [236].$$

This constitutes the *"naïve"* independence assumption made by the algorithm. Although this assumption is rarely true, the classifier nonetheless often works surprisingly well. One reason for this could be the simplicity of the model — with few parameters, it is relatively resistant to overfitting [207].

Furthermore, since $P(x_1, \dots, x_p)$ is constant for the given input, the proportional relationship

$$P(y \mid x_1, \dots, x_p) \propto P(y) \prod_{i=1}^{p} P(x_i \mid y)$$

provides sufficient information to construct a classifier [28]. This is typically achieved by combining the model with a *decision rule*, such as the *maximum a posteriori* (MAP) estimation

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{p} P(x_i|y).$$

The prior probability of the class $y = c$ may be estimated as $\hat{P}(y) = N_c/N$, where N_c is the number of training observations with class label c and N is the total number of training samples [282]. In order to avoid zero probabilities, *smoothing* may be applied such that

$$\hat{P}(y) = \frac{N_c + \alpha}{N + \alpha N}.$$

When $\alpha = 1$, this is referred to as *Laplace* or *add-one smoothing*, whilst *Lidstone smoothing* refers to the case where $\alpha < 1$ [185, 207, 236].

Several variants of the Naïve Bayes classifier exist, each of which differ in their assumptions about $P(x_i \mid y)$. The variants most commonly used include the *Gaussian naïve Bayes* classifier, the *Bernoulli naïve Bayes* classifier and the *multinomial naïve Bayes* classifier [236].

The Gaussian naïve Bayes classifier is typically used when the feature vector $\boldsymbol{x} \in \mathbb{R}^p$ [208]. The likelihood of each of the features is then assumed to be distributed according to a Gaussian distribution [236], such that

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right).$$

By the maximum likelihood estimate, the parameters μ_y and σ_y are equal to the sample mean and sample standard deviation, respectively (calculated separately for each class, using the subset of observations with class label $y = c$) [335].

In cases where $x_i \in \{0, 1\}$, the Bernoulli naïve Bayes classifier is employed. Each element x_i in the feature vector then takes on a binary value, indicating whether or not feature $i \in \{1, \ldots, p\}$ is present in the observation. The likelihood of the presence of each feature is assumed to be distributed according to the binomial distribution, such that

$$P(x_i \mid y) = p_{iy}^{x_i}(1 - p_{iy})^{(1-x_i)},$$

where p_{iy} is the probability that feature i occurs in class y [208, 236]. This value may be estimated as the relative frequency with which feature i is present in class y. For class $y = c$ this is calculated as $\hat{p}_{ic} = N_{ic}/N_c$, where N_{ic} is the number of observations with class label c for which feature i is present, and N_c is defined as before.

If $\boldsymbol{x} \in \mathbb{Z}^p$, for example, where x_i is the frequency with which feature i was observed in the sample, the multinomial naïve Bayes classifier is used. The probability of observing a given input vector $\boldsymbol{x} = \begin{bmatrix} x_1 & \ldots & x_p \end{bmatrix}$ in class y then is

$$P(\boldsymbol{x} \mid y) = \frac{n!}{x_1! \ldots x_p!} \prod_{j=1}^{p} p_{jy}^{x_i}, \tag{3.23}$$

where $n = x_1 + \ldots + x_p$ and p_{jy} is defined and calculated in the same manner as before [282]. Expression (3.23) is often written simply as a proportionality omitting the normalisation term $n!/(x_1! \cdots x_p!)$ [185, 282].

In spite of the oversimplified assumptions of the model, the naïve Bayes classifier has been shown to perform well, especially in the context of text classification and spam filtering problems. Advantages of the model include its computational speed and suitability for sparse data sets of relatively small sizes. Although it is viewed as a "decent" classifier, its performance as a probability estimator is reportedly poor [236].

3.3.5 Logistic regression

Logistic regression is a probabilistic, discriminative classifier, designed to predict the category to which a given observation belongs [30]. Specifically, it is suited to the case of binary classification, where the class label $y = \{0, 1\}$ is to be predicted for an observation $\boldsymbol{x} = \begin{bmatrix} x_1 & \dots & x_p \end{bmatrix}$.

First, a linear transformation is applied to the input data. Subsequently, a scalar value is obtained by summing over p and adding a *bias* term. This value, known as the *score*, is then passed through a non-linearity in the form of the *sigmoid function*, given by $\phi(z) = 1/(1 + e^{-z})$. This function has the desirable property that it is bounded from below by zero and from above by one, rendering it suitable to represent probabilities. The final model may be expressed in the form

$$P(y = 1 \mid \boldsymbol{x}) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p)}}, \tag{3.24}$$

where $\beta_0, \beta_1, \ldots, \beta_p$ are the *coefficients* of the model and β_0 represents the bias [135, 169]. The category is then predicted using a threshold, as described in §3.2.3.

Figure 3.11 illustrates a logistic regression model fitted to data describing the relationship between an individual's propensity to default on a loan and the average outstanding balance on his or her credit card after the corresponding loan instalment is deducted. The dashes in the figure represent the true labels of the training data and the solid line represents the probability of default as predicted by the model.

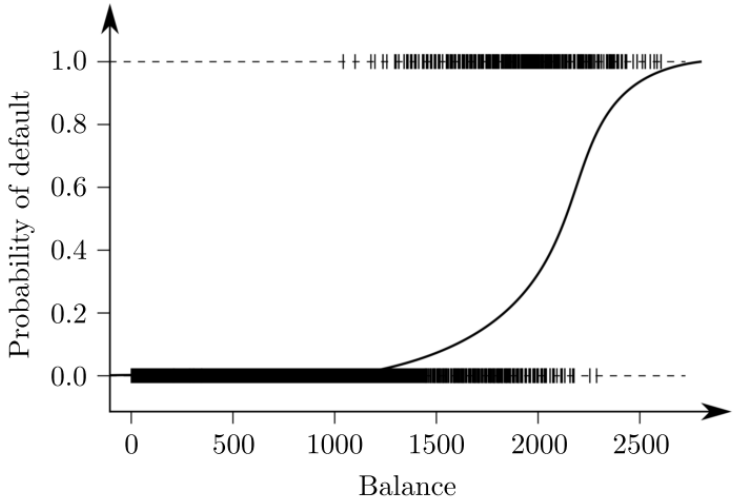


FIGURE 3.11: *A fitted logistic regression model for predicting whether or not an individual will default on a loan based on the average balance remaining on their credit card after the monthly instalment [135].*

In order to estimate the model coefficients, the *maximum likelihood* method is typically applied. Intuitively, this entails choosing coefficients such that (3.24) is maximised when the response measurement of a training sample y_i is equal to one and minimised when $y_i = 0$. Formally, the

likelihood function is defined as

$$\ell(\beta_0, \beta_1, \ldots, \beta_p) = \prod_{i=1}^{n} P(y_i = 1 \mid \boldsymbol{x}_i)^{y_i}(1 - P(y_i = 1 \mid \boldsymbol{x}_i))^{1-y_i},$$

where n is the number of training observations and where every observation in the training data set is viewed as a Bernoulli trial. Taking the logarithm yields

$$\sum_{i=1}^{n} y_i \log(P(y_i = 1 \mid \boldsymbol{x}_i)) + (1 - y_i) \log(1 - P(y_i = 1 \mid \boldsymbol{x}_i)). \tag{3.25}$$

There is no closed-form solution to the problem of maximising (3.25) [212, 169]. Approximate methods are therefore employed, which typically attempt to minimise the negated form of the equation, known as the *cost function*[6]. Typically, gradient-based methods are employed for this purpose, such as those described in §2.2 [169]. The topic of optimisation in the context of cost function minimisation is revisited in the realm of neural networks in a subsequent section.

3.3.6 Maximum entropy

The expression in (3.24) may be simplified by employing the *bias trick* [297]. Instead of writing out the bias term explicitly, an additional element can be added to the input vector $\boldsymbol{x}$, namely $x_{p+1} = 1$. Then the expression can be rewritten as

$$P(y = 1 \mid \boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^T \boldsymbol{x}}} = \pi_1(\boldsymbol{x}),$$

where $\boldsymbol{\theta}$ is a *weight vector* containing the model parameters. Multiplying both the numerator and the denominator by $e^{\boldsymbol{\theta}^T \boldsymbol{x}}$ yields

$$\pi_1(\boldsymbol{x}) = \frac{e^{\boldsymbol{\theta}^T \boldsymbol{x}}}{e^{\boldsymbol{\theta}^T \boldsymbol{x}} + 1}.$$

This function has the desirable properties that $\pi_1(\boldsymbol{x})$ is bounded between zero and one and large if the true class label is equal to one. Since only two classes exist, one can then set $\pi_2(\boldsymbol{x}) = 1 - \pi_1(\boldsymbol{x})$ in order to satisfy the requirement that $\sum_i^k \pi_k(\boldsymbol{x}) = 1$. Unfortunately, however, this does not generalise well beyond binary classification. One may, instead, choose to express both class probabilities as

$$\pi_i(\boldsymbol{x}) = \frac{e^{\boldsymbol{\theta}_i^T \boldsymbol{x}}}{\sum_{j=1}^{k} e^{\boldsymbol{\theta}_j^T \boldsymbol{x}}}.$$

This formulation is similar and features the same properties, but it has the advantage of notational symmetry for $\pi_1(\boldsymbol{x})$ and $\pi_2(\boldsymbol{x})$ and generalises to multi-class classification problems for $k > 2$ [203]. The function $\sigma_i(z) = e^{z_i}/(\sum_{j=1}^{k} e^{z_j})$ for $i = 1, \ldots, k$ is referred to as the *softmax function* [169, 297] and the resulting classifier is called the *maximum entropy* classifier, or multi-class (multinomial) logistic regression. Figure 3.12 contains a schematic representation of the ME classifier for a three-dimensional input vector $\boldsymbol{x} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^T$ and two output classes by means of a *computational graph*[7].

[6] In the literature, the terms *cost function* and *loss function* are often used interchangeably. In this dissertation, the loss function is denoted by L_i and represents the error for a particular observation, whilst the cost function is calculated as $L = \sum_{i=1}^{n} L_i$, where n is the number of training observations.

[7] A computational graph is a representation of a mathematical function as a graph. The *nodes* of the graph are connected by *edges* and represent either input values or mathematical operations.

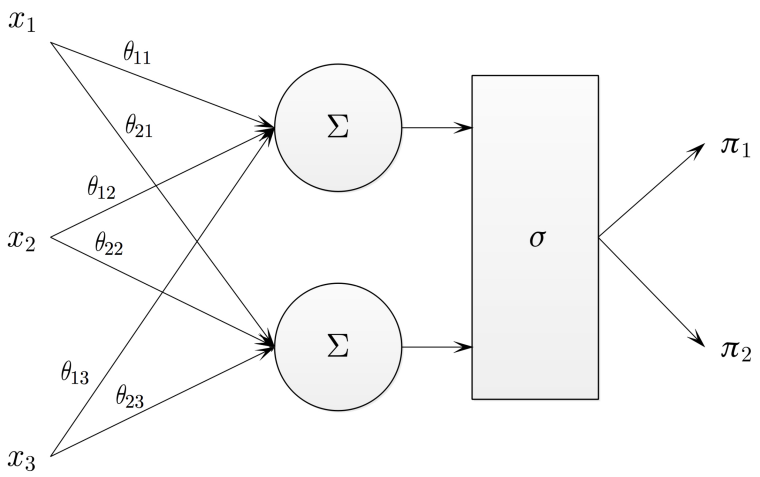


FIGURE 3.12: *An illustration of the maximum entropy classifier for three-dimensional input and $k = 2$ by means of a computational graph. The input is multiplied by the weight vectors $\boldsymbol{\theta}_1 = \begin{bmatrix}\theta_{11} & \theta_{12}\end{bmatrix}$ and $\boldsymbol{\theta}_2 = \begin{bmatrix}\theta_{21} & \theta_{22}\end{bmatrix}$, respectively. The outputs are then summed separately before passing through a softmax function σ, which yields the class probabilities π_1 and π_2. (Note: The bias has been ignored in this case for simplicity.)*

In order to define the likelihood function for the maximum entropy classifier, a 1-of-K encoding scheme is typically used, where the label for each sample is represented by a binary *target vector*, $\boldsymbol{t}_i$, of length k, whose entries are all zero, with the exception of the element corresponding to the true class of the sample (which equals one). For a five-class prediction, for example, the target vector for a sample which belongs to the second class is given by $\boldsymbol{t} = \begin{bmatrix}0 & 1 & 0 & 0 & 0\end{bmatrix}$. The likelihood function for n samples and k classes may then be expressed as

$$p(\boldsymbol{T} \mid \boldsymbol{t}_1, \ldots, \boldsymbol{t}_k) = \prod_{i=1}^{n} \prod_{j=1}^{k} (\pi_j(\boldsymbol{x}_i))^{t_{ij}},$$

where $\boldsymbol{T}$ is an $n \times k$ matrix of target variables containing the element t_{ij} in its i^{th} row and j^{th} column. This entry is equal to one if sample i belongs to class j, or zero otherwise. Taking the negative logarithm yields

$$L(\boldsymbol{t}_1, \ldots, \boldsymbol{t}_k) = -\sum_{i=1}^{n} \sum_{j=1}^{k} t_{ij} \log\big(\pi_j(\boldsymbol{x}_i)\big),$$

which is known as the *cross-entropy cost function*, and is minimised using the same methods that are applicable to logistic regression [30, 203].

3.4 Ensemble learning

An *ensemble* of models is a set of learning models whose individual predictions are combined in some way [74] in order to attain a more generalisable result [361]. The intuition underlying this approach is that different models exhibit different *inductive biases*. If the errors caused by these biases are uncorrelated, the models in the ensemble are expected to compensate for each other's errors, resulting in a decrease in the overall error when the model results are aggregated. It

has been shown that ensemble methods can effectively leverage this property in order to reduce variance error without increasing bias error [260] (see §3.2.2).

The notion that the combination of several *'weak'* classifiers[8] can be combined to form a *'strong'* classifier has been studied in various contexts. French mathematician Nicolas de Condorcet, for instance, established what is now known as *Condorcet's Jury Theorem* [64]. Consider a group of n jurors who wish to reach a binary decision (*e.g.* to charge a defendant as *guilty* or *not guilty*) by majority vote, where each juror has a probability p of making the correct decision and where the probability that the correct verdict is reached by majority vote is P. The theorem states that if $p > 0.5$, then $P > p$. Furthermore, the theorem also states that $P \to 1$ as $n \to \infty$ for all $p > 0.5$. The theorem has two limiting assumptions, namely that the votes should be independent of one another and that there should only be two possible outcomes. Nevertheless, if these conditions hold true, a jury whose members' judgements are only slightly better than a random vote can come to the correct conclusion provided it is sufficiently large [260, 266].

Analogously, English statistician and polymath Sir Francis Galton discovered the phenomenon of *the wisdom of crowds* whilst attending a livestock fair. As part of a contest, several visitors of the fair submitted guesses estimating the weight of an ox. Galton observed that, although no single visitor succeeded in guessing the true weight of the ox (1 198 pounds), the average of all the guesses (1 197 pounds) came remarkably close to this value. As in the the situation described in Condorcet's Jury Theorem, the aggregation of numerous simplistic predictions resulted in an accurate prediction [260]. As before, however, this is not unconditionally true. According to Surowiekie [307], the following four conditions must hold in order for a crowd to be considered *wise*:

(i) *Diversity of opinon*: Each person should have some information that is *private*. This may take several forms, including a unique or unconventional interpretation of the facts.

(ii) *Independence*: The opinion of an individual should not be influenced by the opinion of others.

(iii) *Decentralisation*: People should each have access to a *local* source of knowledge or specialisation.

(iv) *Aggregation*: There should be some means by which individual judgements may be combined to form a collective decision.

In the remainder of this section, ensemble methods are discussed in the context of classification. Important considerations when constructing an ensemble classifier are first outlined, and this is followed by descriptions of three popular ensemble methods, namely *bagging*, *boosting* and *stacking*.

3.4.1 Constructing an ensemble classifier

A typical ensemble architecture comprises several *base learners*, which take as input an instance of the feature vector $\boldsymbol{x}$, and some means of *combining* or aggregating their results to yield a single prediction $\boldsymbol{y}$, as shown in Figure 3.13. Base learners are typically trained in respect of the training data by means of a *base learning algorithm*, which can be a decision tree, an SVM

[8] A classifier that performs only marginally better than random guessing is referred to as a *weak classifier* [260, 362]. A *strong classifier*, on the other hand, can be arbitrarily accurate [260].

or any other algorithm similar to those reviewed in the previous section [362]. Mathematically, an ensemble learning model may be described as a mapping

$$\varphi(\boldsymbol{x}_i) = G(f_1(\boldsymbol{x}_i), f_2(\boldsymbol{x}_i), \ldots, f_k(\boldsymbol{x}_i)) = \hat{\boldsymbol{y}}_i,$$

where G is an aggregation function and $f_1, f_2, \ldots, f_k$ are the base learners [266]. Often, base learners are generated by applying the same base learning algorithm, resulting in *homogeneous* ensembles. *Heterogeneous* ensembles are, however, also possible, in which different learning algorithms are employed to generate different types of base learners [362]. The latter type of ensemble is typically also referred to as a *multiple classifier system* in the literature [260].

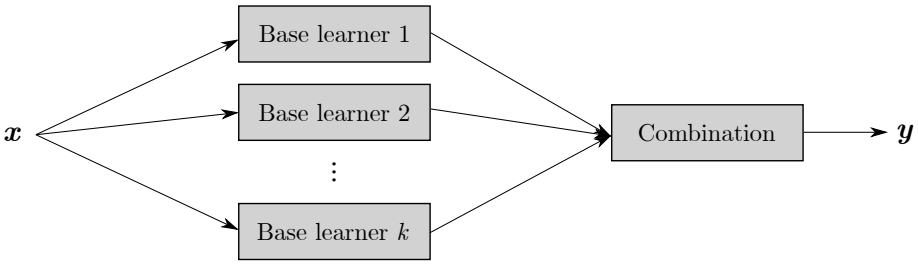
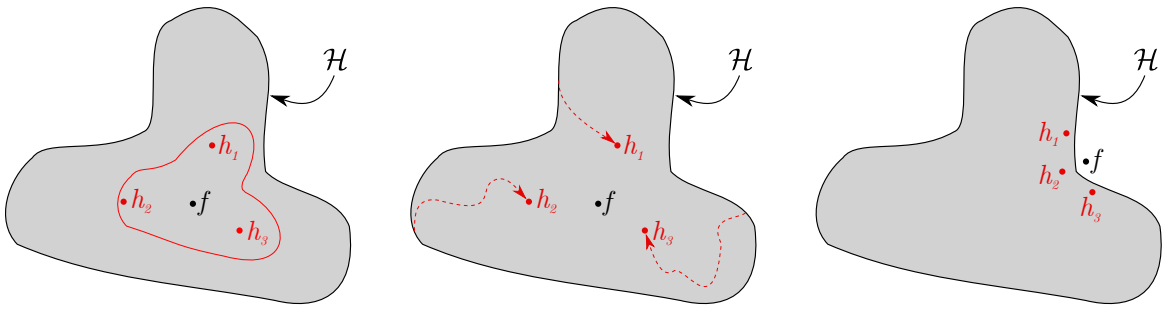


FIGURE 3.13: *A typical ensemble architecture (adapted from Zhou [362]).*

Two pioneering papers on ensemble learning from the year 1990 provided initial evidence that this type of model configuration may result in improved performance. Schapire [273] proved theoretically that a strong classifier can be generated by combining several weak classifiers. Hansen and Salamon [114], on the other hand, showed empirically that the prediction error made by an ensemble of several classifiers was smaller than that of the best single classifier.

Necessary and sufficient conditions for the ensemble to outperform its individual members are that the base learners are both *accurate* (in the sense that they perform better than random guessing) and *diverse* (in the sense that they make different errors on new data points) [74, 114]. Dietterich [74] provided three primary reasons why such an ensemble results in a lower generalisation error. These are illustrated schematically in Figure 3.14.



(a) *Statistical* (b) *Computational* (c) *Representational*

FIGURE 3.14: *Schematic illustrations of the three primary reasons why ensembles may outperform single classifiers (adapted from [74]).*

The first reason is *statistical*. Consider an hypothesis space $\mathcal{H}$ of possible hypotheses describing a set of data. A learning algorithm may be viewed as searching through this space in order to find

the most suitable hypothesis h. If there are too few data points available, the algorithm may find several hypotheses that yield the same accuracy in respect of the training data. By constructing an ensemble of several such accurate classifiers, their hypotheses are effectively averaged, reducing the risk of choosing an incorrect hypothesis and thereby more closely resembling the true hypothesis f, as shown in Figure 3.14(a).

The second reason is *computational*. Many learning algorithms search through the hypothesis space by means of local search algorithms, such as gradient descent (see Algorithm 2.2), which may become stuck in local optima, even when sufficient training data are available. By repeatedly deploying such a search algorithm from different starting points and aggregating the results, a better approximation may be achieved, as shown in Figure 3.14(b).

Finally, ensemble methods may overcome *representational* problems. Often, the true function f is not contained within the hypothesis space $\mathcal{H}$ provided to the algorithm. The aggregation of several different hypotheses in $\mathcal{H}$ may, however, allow for an expansion of the space of representable functions, as shown in Figure 3.14(c).

Diversity

Various different approaches exist for ensuring diversity between base learners. The three most common approaches may be described as follows [74, 266]:

(i) *Input manipulation*: According to this approach, the training algorithm is executed several times, each time employing a different subset of the training data. These subsets can be generated by random sampling, or by partitioning the data set to either include different training examples with all features in each subset (*horizontal partitioning*) or to include all training examples with different groups of features in each subset (*vertical partitioning*).

(ii) *Learning algorithm manipulation*: Diversity may also be induced by altering the manner in which the learning algorithm traverses the hypothesis space by selecting different hyperparameters for each base learner or by introducing randomness into one of the steps of the algorithm. Forming a heterogeneous ensemble comprising different learning algorithms may also be considered a form of learning algorithm manipulation.

(iii) *Output manipulation*: According to this approach, the target variable provided to each learning algorithm is altered. A multi-class ensemble classifier may, for example, be constructed by combining several binary classifiers.

Naturally, it is also possible to combine several of the above approaches when constructing an ensemble. Sagi and Rakoch [266] refer to such a combination as an *ensemble hybridisation approach*.

Dependency

Ensemble methods may also be distinguished by their dependency between base models [266]. Ensembles generated within a *dependent* paradigm are constructed sequentially, allowing the knowledge generated during the training of a previous base learner to influence the construction of the next base learner. In an *independent* paradigm, however, each base learner is built independently of the others. Consequently, this operation may be parallelised. These groups of ensembles are therefore also referred to as *sequential ensemble methods* and *parallel ensemble methods*, respectively [362].

Output aggregation

There are numerous methods for aggregating the output of individual base learners in an ensemble. Most commonly, these may be classified as either *weighting* approaches or *meta-learning* approaches [266, 362].

Weighting approaches include *averaging* and *voting*. The former may be applied to combine quantitative outputs returned by regression algorithms. In *simple averaging*, the arithmetic mean of the predictions of all base learners is returned by the ensemble. This approach is recommended when the performances of the base learners are comparable. In *weighted averaging*, on the other hand, each base learner's prediction is assigned a weight signifying its importance. This weight is often chosen to be proportional to the individual model's strengths and may, for example, represent its relative performance in respect of the validation data set compared to those of the other models in the ensemble. This method is typically recommended when base learners differ significantly in terms of classification accuracy. In general, however, empirical research has not shown weighted averaging to outperform simple averaging [362].

Voting is a method similar to averaging applied to qualitative outputs returned by classification algorithms. The most popular such method is *majority voting*, whereby the ensemble returns the class label predicted by more than half of the base classifiers. In the case where a consensus is not reached by the majority, no prediction is made by the ensemble. In *plurality voting*, on the other hand, the class which receives the largest number of votes is returned by the ensemble regardless of the overall proportion of votes.

The idea behind *weighted voting* methods is to assign a contribution to each classifier that reflects its overall performance, typically as measured in respect of the validation data set, such that the votes of more accurate models are, effectively, counted several times. These weights may be assigned in myriad ways. By adopting a Bayesian approach, it can be shown that the optimal weight w_i of a base learner in an ensemble is proportional to $\log(\alpha_i/(1-\alpha_i))$, where α_i represents the model's accuracy in respect of the validation set. This relationship only holds under the assumption that the outputs of the base learners are independent. Since the base learners are trained in respect of the same problem, however, they are usually "highly" correlated [362]. Opitz and Shavlik [223] therefore suggested setting the weight of a base learner $i \in \{1, 2, \ldots, K\}$ proportional to its validation accuracy as

$$w_i = \frac{\alpha_i}{\sum_{j=1}^{K} \alpha_j},$$

whilst more complex methods employ the *Dempster–Shafer theory of evidence* or the notion of entropy to assign model weights [260]. In practice, finding good weights for the ensemble is a computationally hard problem[9] [362].

In all the voting methods discussed thus far, discrete class labels were assumed. If the ensemble is formed by probabilistic or scoring classifiers, *soft voting* can be employed instead. According to this approach, the probability or score for each class label is averaged across the base learners in the ensemble. A weight may be applied for each classifier, or for each classifier per class, accommodating cases where specific classifiers routinely misclassify instances of certain classes. Soft voting is typically only employed in homogeneous ensembles, since the scores or probabilities returned by models of different types often cannot be compared directly without careful calibration [362].

[9] A problem is considered computationally hard if the asymptotic upper bound of its time complexity is not a polynomial function of the input size.

Meta-learning refers to the process of *'learning from learners'* [266]. More specifically, a two-level training procedure is employed, by which the base learners (or *first-level learners*) are first trained in respect of the training data, and another learning algorithm is employed to train a *second-level learner* in respect of the outputs generated by the first-level learners [362]. This approach works well when different base learners perform better in respect of different subsets of the data [266]. Essentially, the meta-learning approach corresponds to the weighted voting or averaging approach, with the exception that the weights are *learnt* by the second-level learning algorithm rather than being predetermined by some other means.

A popular variant of meta-learning is the *mixture of experts* [260, 266]. According to this approach, the problem space is partitioned into various sub-problems (*e.g.* certain regions of the feature space). A different model is then trained in respect of each of these sub-problems, and is subsequently referred to as the *expert* for that sub-problem. Each time the ensemble is deployed, one of the experts is consulted based on the type of problem at hand.

According to the definitions above, simply weighting the outputs of several different models can be considered an ensemble, since the conditions of a diverse set of base learners and some means of combining their results are satisfied. In the remainder of this section, three other popular ensemble methods, namely *bagging*, *boosting* and *stacking*, are described in more detail with reference to the considerations described above. Subsequently, the concept of *ensemble pruning* is reviewed.

3.4.2 Bagging

The method of bootstrap aggregating or bagging [35] was already introduced in the context of random forests in §3.3.2. According to this method, diversity is achieved by training each base learner in respect of a different subset of the training data (diversity by input manipulation[10]). Bagging is typically implemented by employing base learners of the same type, and is therefore considered an homogeneous ensemble. Furthermore, each model is constructed independently.

In order to provide a large enough data subset to each of the base learners, bootstrap sampling is employed, which entails uniformly sampling with replacement from the original data set until a new data set of the same size as the original data set is constructed, which contains some repeated data points. The outputs of each base learner are then aggregated by means of a simple average or plurality vote.

This process is illustrated schematically in Figure 3.15 in the case of three base learners, where cylindrical shapes represent data stores, the symbols contained within these data stores represent different training examples and where the white circle represents the aggregation of model outputs by means of weighting. As is evident from the figure, the bootstrap samples each contain one or two repeated elements from the original data set and represent different distributions of the same data. In the configuration in the figure, Base learner 3 will likely more accurately differentiate between stars and squares, whilst Base learners 1 and 2 will likely be better at identifying triangles and circles, respectively. The models are combined with the aim of harnessing the models' complementary strengths in order to predict the output vector $\hat{\boldsymbol{y}}$ accurately.

[10] As described in §3.3.2, the random forests technique additionally ensures diversity by learning algorithm manipulation, since it randomly selects a subset of features that may be considered for the split at each level of the decision tree. Random forests therefore represents an ensemble hybridisation approach.

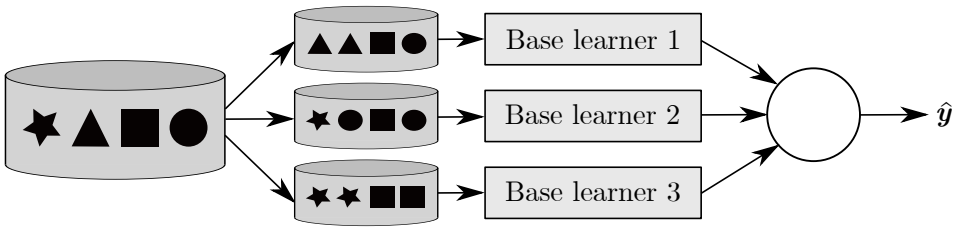


FIGURE 3.15: *A schematic illustration of the bagging ensemble method with three base learners.*

3.4.3 Boosting

Boosting is a procedure by which weak learners may be converted to strong learners, as first outlined by Schapire [273]. As in bagging, diversity is ensured by input manipulation, whereby each base learner is trained in respect of a differently distributed data set. Unlike bagging, however, boosting is a sequential and dependent process. More specifically, the data set for each new base learner is generated to target the errors made by previous base learners. The general boosting procedure is illustrated in Figure 3.16 by means of an example with three base learners.

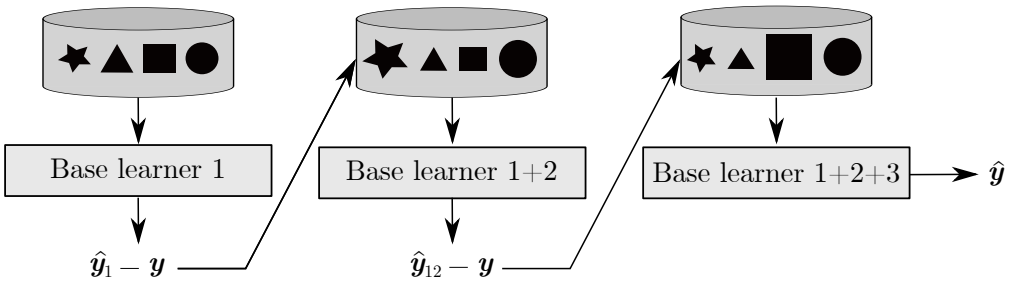


FIGURE 3.16: *A schematic illustration of the boosting ensemble method with three base learners.*

Base learner 1 is trained in respect of the original training data. Subsequently, its error, denoted by the difference between the predicted outcome $\hat{\boldsymbol{y}}_i$ and the true label $\boldsymbol{y}$, is observed. In the case of the example, the model was found to misclassify instances of the star and circle classes. The data distribution is then adjusted to give more weight to those classes that were misclassified, as indicated by the larger shapes in the second data distribution. A second base learner is then trained in respect of the new distribution, and its predictions are weighted with those of Base learner 1 to form an initial two-model ensemble. The results of this ensemble are then employed to guide the next data distribution, and the process is repeated a specified number of times. By iteratively adding new models targeted to address the weaknesses of the previous models, weak learners are *boosted* to form a single strong learner.

The boosting procedure has been adapted by several algorithms, including the award-winning *AdaBoost* [89] algorithm and *XGBoost* [50], which was employed in 17 of the 29 winning solutions to competitions hosted on the popular platform *Kaggle* [260, 266].

3.4.4 Stacking

Stacking [343] is an ensemble learning method in which a meta-learner is trained to combine several independent base learners. This type of ensemble is typically heterogeneous, with diversity stemming from the different learning algorithms employed. This is illustrated in Figure 3.17,

where the same data set is employed to train base learners of different types (denoted by different shades of grey). As is true for the base learners in an ensemble, any learning algorithm can be employed as the meta-learning algorithm. Typically, however, some form of linear or logistic regression (see §3.3.5) is applied [236, 343].

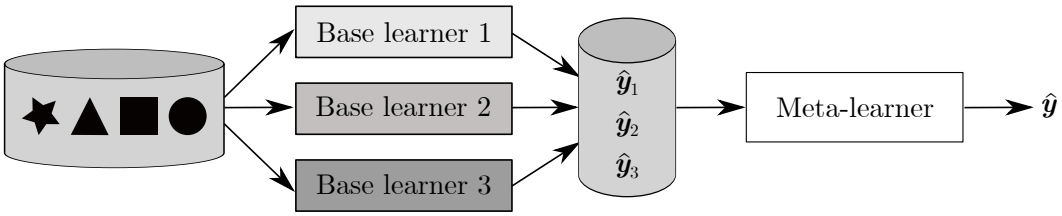


FIGURE 3.17: *A schematic illustration of the stacking ensemble method with three base learners.*

As indicated in the figure, the meta-learning algorithm is trained in respect of a new data set formed by the predicted outputs $\hat{\boldsymbol{y}}_i$ of each of the base learners $i \in \{1, 2, \ldots, K\}$ in order to return a collated prediction $\hat{\boldsymbol{y}}$. If the predictions which form the training data for the meta-learning algorithm were made in respect of the same data employed to train the base learners, there would be a high risk of overfitting [362]. A k-fold cross-validation approach is therefore typically applied, whereby the base learners are trained in respect of $k-1$ folds and the predictions in respect of the k^{th} fold are included in the training data set for the meta-learner. This process is repeated k times, yielding a new training set of the same size as the original data set. The base learners are subsequently re-trained in respect of the entire training data set for use during the deployment phase.

Ting and Witten [314] recommended using the predicted class probabilities of each base learner as input to the meta-learner for classification problems instead of discrete class labels. In this approach, the number of input features is increased from K to $K \times C$, where C is the number of classes in the data set. They, furthermore, proposed formulating the problem as a multi-response regression problem by employing a *one-versus-all* approach. This approach is similar to the one-versus-one approach described in §3.3.3, except that C classifiers are formed to predict the probability with which a data point belongs to class $i \in \{1, 2, \ldots, C\}$ and not to any of the other classes, rather than constructing pairwise comparisons between all classes. Seewald [277] later suggested that only the class probabilities associated with the target class should be employed as input features for each of these one-versus-all classifiers. This variation of the stacking algorithm is referred to as *StackingC* [362].

3.4.5 Ensemble pruning

As shown in Figure 3.13, a typical ensemble comprises two phases: The generation of base learners and their combination. An ensemble comprising a large number of models may, however, be associated with a high computational expense and a large memory requirement. Furthermore, retaining base learners with poor performance may compromise the ensemble's ability to achieve superior results compared to its component learners. Several researchers have, therefore, added an additional intermediate phase with the purpose of reducing the size of the ensemble prior to combination. This phase, commonly referred to as *ensemble pruning* or *ensemble selection*, seeks to improve both the ensemble's efficiency and its predictive performance [317, 338].

The selection of an appropriate set of base learners is a complex task. In the case of three base learners L_1, L_2 and L_3, seven subsets can be generated. These include the set of all three base

learners, the combinations $\{L_1, L_2\}$, $\{L_1, L_3\}$ and $\{L_2, L_3\}$, as well as the three singleton sets each containing one of the base learners. The number of possible subsets that can be formed with N base learners, given by $\sum_{i=1}^{N} \binom{N}{i}$, increases to 31 for $N = 5$, to 1 023 for $N = 10$ and to 1 048 575 for $N = 20$. It therefore quickly becomes infeasible to select the best ensemble configuration by brute force.

Tsoumakas *et al.* [317] categorised ensemble pruning approaches into four categories. The first refers to *search-based methods*, which employ heuristics to search through the space of possible base learner combinations in order to maximise some evaluation criterion. These methods may further be classified as *greedy* or *stochastic*.

Whalen and Pandey [338], for example, employed a simple greedy approach in which the K top performing classifiers are added to the ensemble. They compared this approach with a variation by Caruana *et al.* [46, 47], in which an ensemble is iteratively generated by randomly selecting a small subset of candidate base learners during each iteration and then including the candidate whose addition to the ensemble results in the maximum performance of the ensemble. A notable drawback of such approaches is the requirement that the ensemble size K should be selected beforehand. Furthermore, only the accuracy of the individual base learners is considered in the first approach, without regard for the effect of the base learners' inclusion on the ensemble performance. Several other researchers have employed evolutionary algorithms, such as the *genetic algorithm* [170, 226], *ant colony optimisation* [51], the *artificial bee colony algorithm* [285] and *particle swarm optimisation* [221] to select base learners. The objective functions of these algorithms are typically related to the performance of the entire ensemble. If simple voting is employed as the output aggregation method, this value is easily computed. If, however, a meta-learning algorithm is employed, the approach comes at a significant computationally cost, since training and prediction steps must be performed during each iteration.

Other researchers have, therefore, computed evaluation measures reflecting the diversity and accuracy of a selected subset of base learners, based on the notion that choosing accurate and diverse base learners should result in an accurate ensemble [108, 317]. This approach, however, introduces a different problem, since there is neither a formal definition of model diversity nor a generally accepted means of quantifying its magnitude [108, 278, 317]. Kadkhodaei and Moghadam [141] quantified diversity based on a measure of entropy related to the classification of individual observations by the base learners. More specifically, entropy (or diversity) as defined in the paper is maximised when half the base learners vote for the positive class and the other half vote for the negative class in a binary classification problem. A genetic algorithm was then employed to maximise entropy by selectively including certain base learners in the ensemble. Gu *et al.* [108] presented a survey of multi-objective ensemble selection techniques, where ensemble selection is performed in the context of a trade-off between accuracy and diversity. Several additional measures of diversity may be found in the survey [108], many of which consider classifiers that make different errors in respect of training or validation data as diverse. By this definition, an increase in ensemble accuracy results in a decrease in ensemble diversity, since the overall frequency of errors is reduced. Accuracy and diversity are therefore often viewed as conflicting objectives, and *Pareto optimal*[11] solutions to the ensemble selection problem are typically sought.

Approaches within the second category in the taxonomy by Tsoumakas *et al.* [317], namely *clustering-based methods*, typically employ a two-stage process. First, unsupervised learning algorithms are employed to partition base learners into clusters of models that make similar predictions. Each cluster is then separately pruned in order to increase the overall diversity of

[11] In this context, a solution is *Pareto optimal* if any other solution that results in a higher accuracy (diversity) is also associated with a lower diversity (accuracy).

the ensemble. Giacinto *et al.* [97], for instance, employed *hierarchical agglomerative clustering* to partition classifiers into groups of learners that made similar errors in respect of a validation set. The base learner in a cluster whose predictions are most dissimilar to those of learners from other clusters is then chosen to represent each cluster, whilst majority voting is employed to aggregate predictions from all clusters. The number of clusters is chosen to maximise the validation accuracy of the pruned ensemble.

The third category comprises *ranking-based methods*, which sort candidate base learners according to some evaluation metric and then select a predefined number of base learners according to this sorting order. In *orientation ordering*, for example, a *signature vector* is computed for each base learner. The signature vector is a binary vector with one entry for each training observation, taking the value one if the observation is correctly classified, or zero otherwise. The *ensemble signature vector* is the average of all base learner signature vectors and represents the voting ensemble's predictive accuracy. Furthermore, the *reference vector* is chosen as a vector perpendicular to the ensemble signature vector. The base learners are, finally, ordered according to the angle between their signature vector and the reference vector. Only those base learners whose angle is less than $\pi/2$ radians are included in the ensemble. In essence, this method gives preference to base learners that correctly classify instances misclassified by the full ensemble. Rooney *et al.* [262], on the other hand, defined a composite metric $accuracy_i + \alpha(diversity_i)$ for each candidate base learner i. This metric is composed of the base learner's validation accuracy (relative to that of the most accurate base learner) and the base learner's contribution to overall ensemble diversity, weighted by a user-specified parameter $\alpha \in [0, 1]$. Two different measures of diversity were compared in the paper. The first represents the proportion of models in the ensemble to which the base learner's prediction error is not highly correlated[12]. The second represents the normalised variance of the base learner's predictions relative to the average predictions of all candidate base learners. The K base learners with the highest composite metrics were then included in the ensemble.

Finally, Tsoumakas *et al.* [317] defined a fourth category to accommodate all *other* approaches to ensemble pruning. This includes, for example, a statistical approach by which only classifiers whose performances are significantly[13] better than those of the other base learners are retained in the ensemble.

Onan [221] compared the relative performances of various homogeneous and heterogeneous ensemble models, as well as several evolutionary approaches towards ensemble selection in combination with stacking, in respect of text classification problems. They found that heterogeneous ensembles (*i.e.* stacking) outperformed homogeneous ensembles (*e.g.* AdaBoost, Bagging and Random Forest), whilst stacking configurations optimised by means of evolutionary algorithms performed better than regular stacking methods. Whalen and Pandey [338] found that a voting ensemble with base learners selected according to a greedy approach matched or outperformed a stacking classifier using all base learners in respect of four different data sets. The stochastic iterative ensemble selection method of Caruana *et al.* [46, 47] consistently outperformed the greedy approach, whilst a combination of cluster-based ensemble selection and stacking performed comparably with or marginally worse than the greedy approach. The best results were, however, consistently achieved by first aggregating the results of base learners with the same learning algorithm and then stacking the aggregated results. This may indicate that selecting base learners based on the diversity of their underlying learning algorithms may present a promising approach.

[12] Rooney *et al.* [262] considered errors with a Pearson correlation coefficient [7] of 0.6 or greater as highly correlated.

[13] A difference between two values is considered statistically significant if it is unlikely that a difference of such magnitude would be observed due to sampling error alone.

3.5 Deep learning

An *artificial neural network* (ANN) is, in essence, a nesting of functions of the form $g(f(x))$, where $f(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$ is a linear *score function* and $g(s)$ is a non-linear *activation function*, such as $\tanh(s)$ or $\max\{0, s\}$. The terms $\boldsymbol{W}$ and $\boldsymbol{b}$ in the scoring function are the *weight matrix* and *bias vector*, respectively [169, 358]. Neural networks are typically represented in the form of a computational graph, with *compute nodes* organised in *layers* and connected by edges in a directional manner [201]. An example of such a representation is shown in Figure 3.18. The neurons in an ANN are loosely modelled on those found in the human brain, which process signals from surrounding neurons (modelled by the weighted sum of inputs) and pass on signals if certain thresholds are exceeded (non-linear activation) [169].

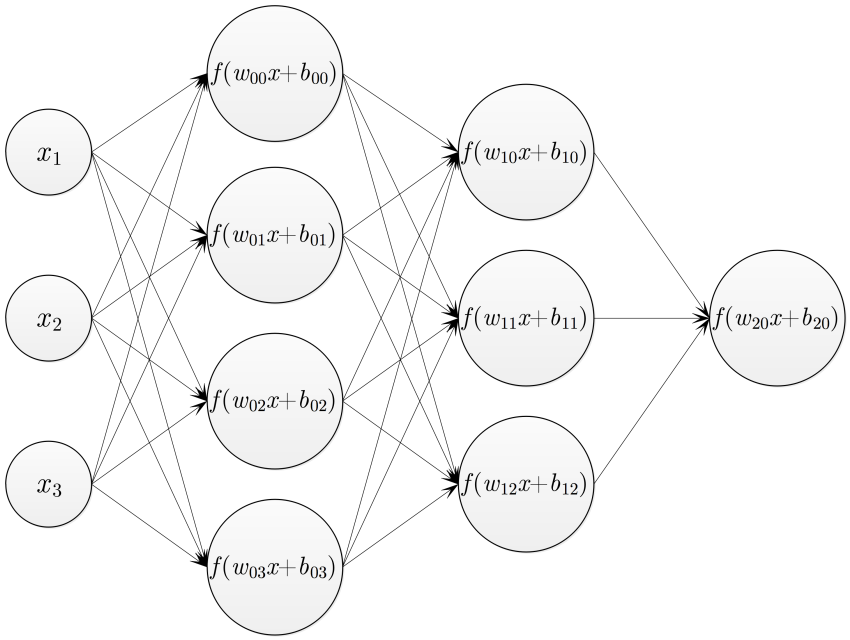
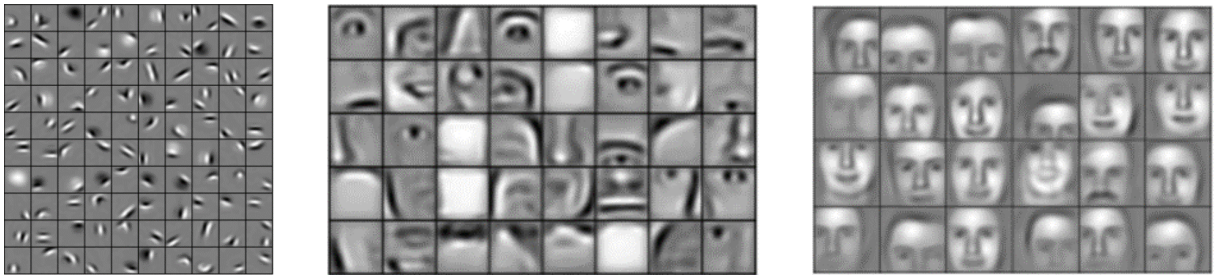


FIGURE 3.18: *A feedforward ANN with two hidden layers. The x in each of the function arguments represents the output of the previous layer.*

The network shown in the figure is referred to as a *feedforward* neural network, since the edges of the graph flow in a single direction. The layers in the graph are vertically organised, where the *input layer* is given by $\boldsymbol{x} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}$, the *output layer* is the node $f(\boldsymbol{w}_{20}\boldsymbol{x} + \boldsymbol{b}_{20})$ and the remaining layers are referred to as the *hidden layers* of the network. The output of any given layer is taken as the input $\boldsymbol{x}$ for a subsequent layer. An alternative formulation for this network in matrix notation is given by $f(\boldsymbol{W}_2 f(\boldsymbol{W}_1 f(\boldsymbol{W}_0\boldsymbol{x} + \boldsymbol{b}_0) + \boldsymbol{b}_1) + \boldsymbol{b}_2)$, where f is any chosen activation function.

Deep learning refers to the application of ANNs to learning tasks, where a network comprises several hidden layers [358]. The central idea in this machine learning paradigm is that, through the nesting of non-linear functions, arbitrarily complex relationships can be modelled to solve a variety of problems. Neural networks are considered *universal approximators*, which means that, given enough hidden layers, they are able to model any sufficiently smooth function to any desired level of accuracy [207]. A significant advantage of the deep learning approach over the machine learning algorithms described in the previous section is a shift of the responsibility for *feature extraction* from the modeller to the model. By visualising the weight vectors learnt by a

neural network that is applied in image recognition tasks, for example, it can be observed that the network extracts certain low-level features, such as edges and colour contrasts, in the first layers and iteratively combines these features to create higher-level features that are meaningful for solving the task at hand. An illustration of this phenomenon is shown in Figure 3.19, which visualises features extracted by an ANN trained for facial recognition at various levels.



(a) *Low-level features* (b) *Mid-level features* (c) *High-level features*

FIGURE 3.19: *Visualisations of the features extracted by a neural network for a facial recognition task [171].*

There is a considerable amount of freedom in the way that ANNs can be designed and trained, with no restrictions on, for example, the number of neurons or layers employed, the number of edges that exist between nodes or the manner in which network parameters are found. There are, however, certain approaches that have proven to be effective in many cases. In the remainder of this section, typically used techniques for optimising the weights of an ANN are described, and this is followed by a documentation of important considerations for the design choices made with respect to network components. Finally, common types of ANNs are reviewed.

3.5.1 Training neural networks

The parameters of an ANN include the weights and biases of each layer, which can be summarised in a parameter matrix $\boldsymbol{\theta}$. The values of these parameters may be found by minimising the cost function $L = \sum_{i=1}^{n} L_i(\boldsymbol{\theta}, \boldsymbol{x}_i, \boldsymbol{y}_i)$.

The cost function of the network is typically non-convex. Approximate methods are therefore employed during the optimisation process, as in the case of logistic regression. The most widely used approach is the use of gradient-based optimisers. To this end, the gradient of the loss function with respect to $\boldsymbol{\theta}$, $\nabla_{\boldsymbol{\theta}} L$, is computed by means of *backpropagation* [358]. This procedure is adopted when the analytical gradient is too complicated to compute and entails decomposing the function into easily differentiable computational units and iteratively applying the *chain rule* in order to find the total gradient [169].

Consider an example network where the estimated output $y_i = \max\{0, \boldsymbol{\theta}^T \boldsymbol{x}\}$[14] and the ℓ_2 *loss* is applied to compare the error between y_i and the target value, t_i. This computation may be summarised by means of a computational graph, as shown in Figure 3.20.

The partial derivative of the loss function with respect to the parameter w_1 may then be expressed as a product of partial derivatives which are easier to compute. More specifically,

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_i} \frac{\partial y_i}{\partial w_1}.$$

[14] Because the output layer only comprises one neuron, $\boldsymbol{\theta}$ is a vector rather than a matrix in this case.

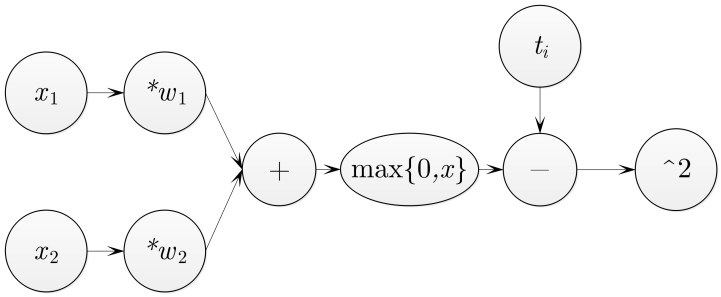


FIGURE 3.20: *An example of a computational graph.*

Using the chain rule, the first term may be calculated as

$$\frac{\partial L}{\partial y_i} = 2(y_i - t_i).$$

Furthermore,

$$\frac{\partial y_i}{\partial w_1} = \begin{cases} x_1 & \text{if } x_1 w_1 > 0, \\ 0 & \text{if } x_1 w_1 \leq 0. \end{cases}$$

This procedure may be extended to arbitrarily complex networks in order to compute the gradient of the loss function. Backpropagation is a standard component of most software packages suitable for deep learning and is implemented automatically in these frameworks [21].

In the remainder of this section, gradient-based optimisation algorithms that are typically employed to train deep neural networks are described, followed by an outline of common regularisation techniques used to combat overfitting during model training.

Gradient descent

Standard application of the gradient descent algorithm described in §2.2 to neural networks is often referred to as *"vanilla" gradient descent*. The basic procedure is outlined in Algorithm 3.2.

Algorithm 3.2: Gradient descent for neural networks

Input : The loss function $L_i(\boldsymbol{\theta}, \boldsymbol{x}_i, \boldsymbol{y}_i)$, the learning rate[15] α and n training samples.
Output: A set of best model parameters, $\boldsymbol{\theta}^*$.

Initialise $\boldsymbol{\theta}^1$ with random values
$k \leftarrow 1$
while *stopping criterion not met* **do**
    $\boldsymbol{\theta}^{k+1} \leftarrow \boldsymbol{\theta}^k - \alpha \nabla_{\boldsymbol{\theta}} \Big(\sum_{i=1}^{n} L_i(\boldsymbol{\theta}, \boldsymbol{x}_i, \boldsymbol{y}_i) \Big)$;
    $k \leftarrow k + 1$;
return $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}^k$

This approach does not guarantee that a global optimum will be found. It turns out, however, that the performances of most local optima are similar in the realm of deep neural networks. Consequently, this is not a critical concern in practice [103]. The computational cost incurred

[15]The term *learning rate* is used to refer to the step size in the gradient descent algorithm within the machine learning community. This terminology is henceforth adopted.

by this method, on the other hand, does pose a problem. Using Algorithm 3.2 requires that the gradient be computed using all observations in the training data in every iteration. Since some networks comprise millions of parameters, this is computationally expensive. An alternative approach is to *estimate* the gradient using a smaller sample of data. This method is referred to as *stochastic gradient descent* (SGD).

Stochastic gradient descent

SGD estimates the gradient of the loss function by employing *mini-batches* consisting of m observations from the training data (yielding n/m batches). The algorithmic procedure is the same as before, except that the gradient is averaged over these m batched samples instead of over the entire set of training data. The batch size m is a hyperparameter. It is typically chosen to be as large as possible, given the performance capabilities of the computer at hand, in order to obtain a more stable estimate of the gradient. Furthermore, its value is often set to equal a power of two in order to make optimal use of the computer's *graphics processing unit* (GPU) [169].

Processing one batch is then referred to as one *iteration* of the algorithm, whilst a complete pass through the training data is referred to as an *epoch*. According to the Robbins-Monroe conditions, SGD is guaranteed to converge to a solution if $\sum_{i=1}^{n} \alpha_i = \infty$ and $\sum_{i=1}^{n} \alpha_i^2 < \infty$ [259].

While it is computationally more efficient than "vanilla" gradient descent, SGD has a significant drawback in that the gradient is scaled equally across all dimensions or features of the network. Consider the contour plot of a convex function to be minimised in Figure 3.21. The red path denotes gradient update steps taken by the algorithm in search of the global minimum represented by the green dot. It is evident from the figure that greater steps should be taken in the horizontal direction whilst steps in the vertical direction should be kept small. Since the same learning rate is adopted in both dimensions, however, this is not possible using SGD, resulting in unnecessarily slow convergence [103].

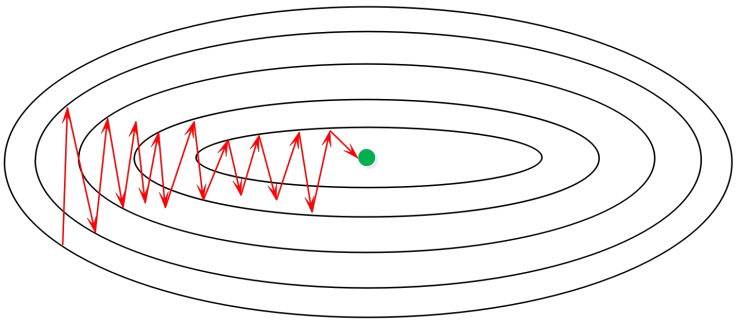


FIGURE 3.21: *The problem of equal scaling in SGD. A contour plot of a function is shown with the global minimum in the centre represented by a green dot. The red arrows denote the search path taken by the SGD algorithm and illustrate its slow convergence when, instead, unequal step sizes are desired for different dimensions.*

Stochastic gradient descent with momentum

SGD with *momentum* [241] is an optimisation algorithm that attempts to address the problem illustrated in Figure 3.21 by introducing a measure of *'velocity'* instead of the gradient. This entity is calculated as

$$\boldsymbol{v}^{k+1} = \beta \boldsymbol{v}^k + \nabla_{\boldsymbol{\theta}} \Big(\sum_{i=1}^{n} L_i(\boldsymbol{\theta}^k, \boldsymbol{x}_i, \boldsymbol{y}_i) \Big), \tag{3.26}$$

where β is known as the *accumulation rate*, *friction coefficient* or *momentum coefficient* [169]. The velocity along a certain direction then increases if the gradient vector repeatedly points in this direction. If, however, the gradient vector points in opposite directions along a certain dimension in an alternating fashion, the velocity in this direction is cancelled out. The parameters of the network are then updated as

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k - \alpha \boldsymbol{v}^{k+1}. \tag{3.27}$$

The hyperparameter β controls the importance of recent gradients relative to gradients computed in previous time steps. When $\beta < 1$, the most recent gradients are weighted more heavily. Typically, β is chosen as 0.9 [264].

The momentum update successfully mitigates the problem of equal scaling associated with the standard SGD and therefore typically converges faster. There is, however, a new problem that arises with this approach. If the velocity in a certain direction is too large, the algorithm may *overshoot* the minimum point. In order to avoid this, an alternative method, called *Nesterov's Momentum*, incorporates a strategy using "*look-ahead momentum*" [24]. Instead of updating the velocity using the gradient at the current point, the current velocity is first used to update the weights to a obtain $\tilde{\boldsymbol{\theta}}^{k+1}$, after which the computations in (3.26)–(3.27) are executed from this new point. The advantage of this approach is that a case of overshooting in the first step can be corrected using the standard momentum approach.

Root-Mean-Squared Propagation

An alternative solution to the problem illustrated in Figure 3.21 has been proposed in the so-called *Root-Mean-Squared Propagation* (RMS Prop) method. Rather than building up velocity in promising directions, this approach seeks to scale down directions of uncertainty. More specifically, the learning rate is divided by an exponentially decaying average of squared gradients (the uncentered variance or *second momentum* of the loss function), given by

$$\boldsymbol{s}^{k+1} = \beta \boldsymbol{s}^k + (1-\beta)\big[\nabla_{\boldsymbol{\theta}} L \otimes \nabla_{\boldsymbol{\theta}} L\big], \tag{3.28}$$

such that

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k - \alpha \frac{\nabla_{\boldsymbol{\theta}} L}{\sqrt{\boldsymbol{s}^{k+1}} + \epsilon},$$

where $\otimes$ represents element-wise multiplication, $\nabla_{\boldsymbol{\theta}} L$ is the cost function of the current mini-batch, and β and ϵ are hyperparameters, typically chosen as 0.9 and 10^{-9}, respectively [169]. This approach effectively dampens oscillations in high variance directions and curbs overshooting, allowing for the selection of a faster learning rate.

Adaptive Movement Estimation

The *Adaptive Movement Estimation* (ADAM) approach [159] combines the momentum and RMS Prop methods to form the most popular and, often, most effective optimisation scheme for neural networks. The update step in ADAM is given by

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k - \alpha \frac{\boldsymbol{m}^{k+1}}{\sqrt{\boldsymbol{v}^{k+1}} + \epsilon},$$

where $\boldsymbol{s}^{k+1}$ is a momentum term, calculated as in (3.26), and $\boldsymbol{v}^{k+1}$ is a variance term, calculated as in (3.28). The hyperparameters used in these calculations are distinguished as β_1 and β_2 for the momentum and variance terms, respectively.

Typically, the momentum and variance terms are initialised to zero. In order to reduce the bias toward zero that is consequently induced, a bias-corrected update is often employed in the forms $\hat{\boldsymbol{m}}^{k+1} = \boldsymbol{m}^{k+1}/(1-\beta_1)$ and $\hat{\boldsymbol{v}}^{k+1} = \boldsymbol{v}^{k+1}/(1-\beta_2)$ [169].

Monitoring the learning process

In order to evaluate whether the learning process is effective, several metrics can be tracked during model training. Most notably, these include the average loss (value of the cost function), validation accuracy and training accuracy of the network during training, as a function of the number of epochs over which the model has been trained [169, 297].

The shape of the loss function over time indicates whether the parameter optimisation process is working (whether the loss is decreasing) as well as whether the learning rate chosen to train the model is suitable.

Given the large scale of neural networks and the millions of parameters typically associated with them, it is often not practical to determine the learning rate by means of the line search method described in §2.2. Instead, the learning rate is treated as a hyperparameter. The importance of this hyperparameter for the performance of the algorithm is shown in Figure 3.22. If the learning rate is too low, the learning process is lengthy since only linear improvements are made using the small step size. If the learning rate is increased, the shape of the curve becomes exponential, resulting in faster training. A learning rate that is too high, however, causes the loss to drop off sharply in the beginning, but converge to a less desirable value. This is due to the fact that the parameters are unable to settle on good values and instead "bounce around chaotically" when the adjustments result in large changes. If update steps are excessively large, the loss begins to diverge [297].

Plotting the training and validation accuracies provides insight into whether or not the model is underfitting or overfitting to the data. Furthermore, the validation accuracy is used in the process of hyperparameter tuning, as described in §3.2.1. A typical such plot is shown in Figure 3.23. As is evident from the figure, the training accuracy continues to increase with subsequent training iterations, whilst the validation accuracy begins to decrease after a certain number of epochs, when the model begins to overfit the data. Prior to this point, the model is in the *underfitting zone* [169]. The difference between the training and validation accuracy is called the *generalisation gap* [268].

There are several methods that can be used to curb overfitting and minimise the generalisation gap, including *early stopping*, *data augmentation*, introducing *parameter norm penalties*, and employing bagging and ensemble methods. Each of these is explained in more detail in the following section.

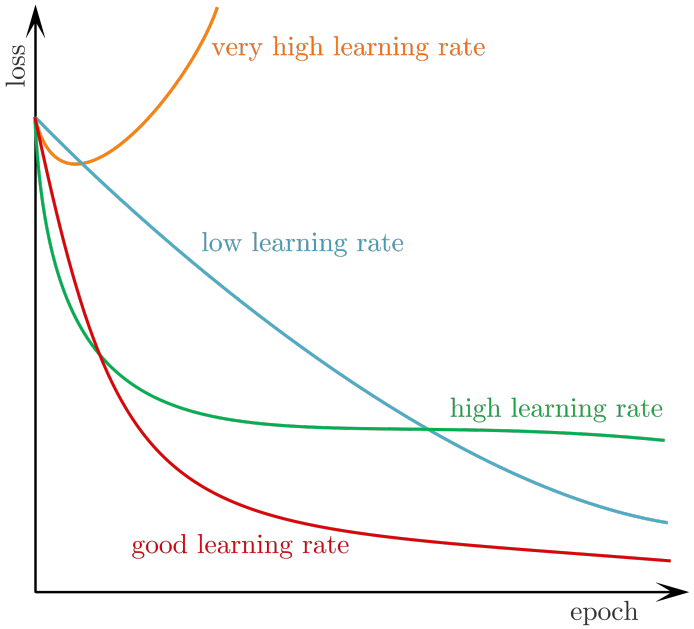


FIGURE 3.22: *The influence of the size of the learning rate on the optimisation process. Low learning rates result in linear improvements, whilst higher learning rates cause exponential improvements. If the learning rate becomes too high, the loss function may diverge, or else converge to an inferior function value [297].*

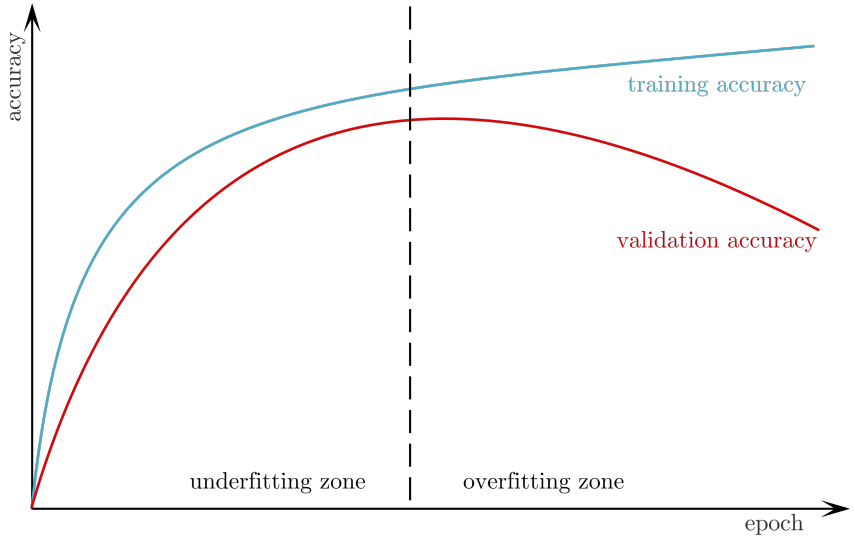


FIGURE 3.23: *The training and validation accuracies are shown as functions of the number of training epochs. The training accuracy increases continuously whilst the validation accuracy begins to decrease after a certain number of epochs. The difference between these accuracy measures is called the generalisation gap.*

Regularisation

Regularisation refers to any strategy that is designed to lower the validation or test error, at the possible expense of increasing the training error. These strategies seek to curb overfitting and improve the generalisability of the model [103]. One of the simplest of these methods is early stopping. In this approach, the validation error is observed throughout the training process and

the point identified at which the validation error begins to decline. The settings of the network parameters prior to this point are then selected for the final model [248].

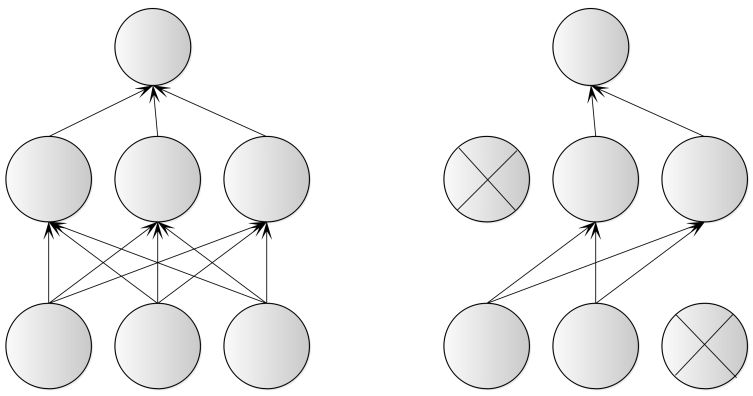
Another approach is to *augment* the training data in an attempt to simulate the wide variety of transformations that may be found in the validation and test sets. In the context of image classification, for example, training images may be cropped, flipped or altered in terms of the brightness and contrast of the image so as to render the classifier invariant to different poses, occlusions and changes in illumination [169].

Perhaps the most widely used forms of regularisation involves the introduction of parameter norm penalties. A *regularisation term* $\lambda R(\boldsymbol{\theta})$ is added to the cost function in order to constrain the values of the weights. Typically, ℓ_1 or ℓ_2 regularisation are used, which add the ℓ_1 and ℓ_2 norms of the weight matrix to the loss function, respectively [103].

The former is based on the principle that the model is more easily generalisable if it has small, distributed weights and all neurons participate in the decision process. The squared term in the cost function penalises large weights, resulting in smaller, more balanced weights throughout the network. This method is therefore also referred to as *weight decay*. The adoption of ℓ_1 regularisation, on the other hand, typically results in some weights being set to zero and others consequently having larger values. By deactivating some neurons (those with a weight of zero), the model capacity is effectively reduced, leading to a simpler model [103, 169].

Finally, bagging and ensemble methods can be used to improve the validation accuracy. As is the case of bagging in random forests, described in §3.3.2, several neural networks can be trained and their results aggregated. If the errors of the models are uncorrelated, the overall error is expected to decrease [35].

Dropout [294] is a technique which employs the concept of an ensemble model in a single neural network. During each training iteration, a random subset of neurons in the network is disabled, as illustrated in Figure 3.24. The set of active neurons during an iteration then form a different model, with a reduced capacity. The collection of models thus formed, each emerging during a different iteration, can then be seen as an ensemble of networks with shared parameters.



(a) *Without dropout* (b) *With dropout*

FIGURE 3.24: *Illustration of the deactivation of neurons upon the application of dropout.*

The goal of dropout is to reduce *co-adaptation* between neurons. Intuitively, neurons will co-adapt by observing that a neighbouring neuron is extracting a certain feature and consequently

'choosing' to focus on a different feature. If a given neuron is exposed to situations during training where a surrounding neuron is deactivated, however, it may start to *'learn'* to identify the same features that this neighbouring neuron had identified previously, in order to maintain the model's performance. The result is a model with redundant representations that is more robust to changes in the input data [169].

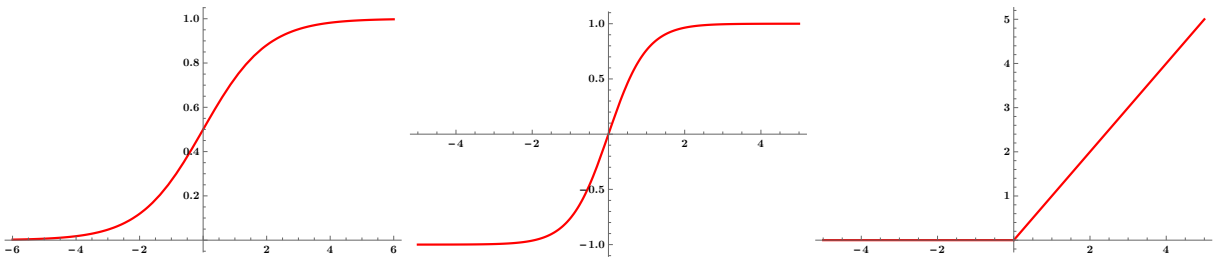
If a neuron was deactivated with a *dropout probability* of p during training, the expected number of neurons active in any given training iteration is $(1-p)N$, where N is the total number of neurons in the network. Consider the case where $N = 2$, and the model output is given by $z = \theta_1 x + \theta_2 y$. If $p = 0.5$, there are four possible cases during training, namely that both neurons are active, that both neurons are inactive, that neuron one is active and neuron two is inactive, or that neuron one is inactive and neuron two is active. The expected value of the output is therefore $E(z) = (1/4)[(\theta_1 x + \theta_2 y) + (0+0) + (\theta_1 x + 0) + (0 + \theta_2 y)] = (1/4)[\theta_1 x + \theta_2 y]$. The output during testing would, therefore, be four times larger than that seen by the optimisation algorithm during training. In order to rectify this problem, the *weight scaling inference rule* may be applied, according to which all weights are multiplied by p during testing [103].

3.5.2 Considerations for network design

As neural networks become deeper, the application of the chain rule results in an increasingly long sequence of multiplications. When individual gradients are significantly small, their product diminishes to a point where the gradient descent algorithm is no longer effective. This phenomenon is referred to as the problem of *vanishing gradients*. Choices made during the design of the network can have a considerable impact on this problem. These include the activation functions that are employed in the hidden layers of the network, the *output function*, which is applied to the output layer of the network and results in the final prediction, as well as the *loss function*, which is used to train the model. Furthermore, the initialisation of network weights and the preprocessing of input data may also have a significant influence [297, 169].

Activation functions

Common choices for activation functions are the sigmoid function, the *hyperbolic tangent* (tanh) function and the *rectified linear unit* (ReLU) function [358]. Plots of the typical shapes of these functions are shown in Figure 3.25.



(a) *Sigmoid* (b) *Hyperbolic tangent* (c) *Rectified linear unit*

FIGURE 3.25: *Plots of typical activation functions used in neural networks.*

The sigmoid function is smooth and differentiable with an output conveniently bounded between zero and one, so that it may be interpreted as a probability. There are, however, two significant drawbacks associated with the sigmoid as an activation function. First, the function only has

a steady gradient when x is close to zero. When $|x|$ becomes large, the function *saturates* (it flattens out), as illustrated in Figure 3.25(a), and the gradients become small. This function is therefore sensitive to initialisation and prone to the problem of vanishing gradients. Secondly, the function value, and consequently the output value of a neuron, is always positive. For a subsequent layer, then, the input is always positive. The gradient vector with respect to the weights therefore necessarily has either all positive or all negative entries (depending on the activation function used in this later layer) according to the chain rule, which constrains the possible directions along which the weight vectors may be optimised, resulting in undesirable *"zig-zag"* patterns [169, 358].

The hyperbolic tangent is zero-centred and therefore does not suffer from the problem of positive output. As is evident from Figure 3.25(b), however, this function also saturates when the absolute value of the input grows large.

The rectified linear unit has recently become popular in neural networks and is now the standard choice for activation functions [168]. It is easy to compute and features large and consistent gradients. Furthermore, it is easy to compute and does not saturate. Upon examining the graph in Figure 3.25(c), however, it is clear that the gradient not only saturates but dies out completely when the input is negative, leading to a phenomenon known as *"dead ReLU."* In such a case, the neuron will always output zero during future iterations, regardless of the input it receives. This problem can be alleviated by initialising the data with a slightly positive bias, or by adopting a variant of the ReLU function, such as the *"leaky" ReLU*, $f(x) = \max\{0.01x, x\}$. In practice, however, this issue does not seem to have a negative effect on network performance [297, 169]. Networks employing ReLUs typically outperform or match the performance of networks using the *tanh* function [358].

Another alternative for activation functions are *maxout units*, which, instead of taking the maximum value of x and zero, computes $\max\{\boldsymbol{w}_1\boldsymbol{x} + \boldsymbol{b}_1, \boldsymbol{w}_2\boldsymbol{x} + \boldsymbol{b}_2\}$, effectively doubling the number of parameters in the network. This is a generalisation of ReLUs and can be used to construct piecewise linear approximations of convex functions with linear regimes that neither die nor saturate. In practice, however, it is considered to be more effective to add more layers to the network than parameters to layers [169].

Output and loss functions

The last layer of a neural network employed for classification purposes is typically a single neuron with a sigmoid activation function (for binary classification), or a layer with as many neurons as classes passed through a softmax function (for non-binary classification) [169, 358]. A notable parallel can be drawn with the logistic regression and maximum entropy classifiers. These models are, in fact, neural networks consisting solely of an input and an output layer [169].

The purpose of the loss function is to assign some numerical value to the classification error made by the network. One may consider employing losses derived form the ℓ_1 or ℓ_2 norms, which compute the distance between the predicted output and the actual output. The ℓ_1 *loss*, given by

$$L_i^1 = \sum_{i=1}^{n} |y_i - f(x_i)|,$$

where y_i is the actual class label for sample i and $f(x_i)$ is the predicted class label, is known to be costly to compute from a computational perspective [274, 169]. The ℓ_2 loss is given by

$$L_i^2 = (y_i - f(x_i))^2.$$

This function is easier to differentiate and optimise, but it is sensitive to outliers due to the squared error term and therefore more prone to the problem of vanishing or *exploding gradients*[16] [169, 359]. These loss functions are collectively referred to as *naïve losses* and are not the popular choice among the deep learning community [297, 169].

The preferred alternatives are the *cross entropy loss* and the *hinge loss*. The former was previously discussed in the context of logistic regression and is given by

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right),$$

where s_{y_i} is the *score* of the true class of the sample. This loss function therefore continuously strives to set the score of the correct class equal to one. The *hinge loss*, on the other hand, saturates when the score for a given class is "good enough" for classification. The formulation

$$L_i = \sum_{j \neq i} \max\{0, s_j - s_{y_i} + 1\}$$

ensures that the loss for a particular sample is equal to zero if the score (before the application of the softmax function) of the correct class is greater than the scores of all other classes by at least one. This saturation can lead to a more balanced output for the classifier, since gradient updates are continued only for classes where the classifier is performing poorly, until a similar level of performance is achieved for all classes [169].

Data pre-processing and weight initialisation

Although much of the *feature engineering* workload is transferred to the model itself in the deep learning context, neural networks, like most machine learning models, perform best when the input data are zero-centred and normalised to have an equal variance of one [169].

Another important consideration is the initialisation process of the weight parameters of the network. One option is to initialise all weights to zero before training. This, however, causes all neurons in the network to compute the same function values. Consequently, the gradients are also equal throughout the network. Since no *symmetry breaking* is achieved, most neurons in the network are then redundant.

Alternatively, one could initialise all weights to random numbers in order to achieve symmetry breaking. If these random numbers are too small, however, the activations "die out" along the network, leading to the problem of vanishing gradients. On the other hand, random numbers that are too large typically cause the data flowing through the network to fall into the saturated regions of the activation functions (causing vanishing gradients) or result in unstable output [297, 169].

In order to solve these problems, one could either further improve the initialisation process or scale the output of each hidden layer to the desired range. Both approaches are based on the premise that the desired output for any neuron is likely normally distributed with a mean of zero and a unit variance [169].

[16] Whilst the problem of *vanishing gradients* refers to the situation in which the gradients tend exponentially fast to zero, the problem of *exploding gradients* is the opposite case, where the gradients exhibit large increases during training [235].

The former can be achieved using the *Xavier initialisation* process. The variance of the score computed in a given neuron, $\text{var}(s) = \text{var}(\sum_{i=1}^{n} w_i x_i) = \sum_{i=1}^{n} \text{var}(w_i x_i)$, can be expanded to

$$\text{var}(s) = \sum_{i=1}^{n} E(w_i)^2 \text{var}(x_i) + E(x_i)^2 \text{var}(w_i) + \text{var}(x_i)\text{var}(w_i),$$

assuming that the samples are independent. Since both the input and the weight vectors have a mean of zero, their expected value is also zero, yielding $\text{var}(s) = \sum_{i=1}^{n} \text{var}(x_i)\text{var}(w_i)$. Under the assumption that the samples are identically distributed, this can be written as $\text{var}(s) = (n\text{var}(x_i))\text{var}(w_i)$. In order to ensure that the variance of the output s equals that of the input x, one should, therefore, set $\text{var}(w) = 1/n$ [99]. In the case of the ReLU function, half of the neurons fall into the "dead ReLU" region when the data are zero-centred. To reduce the number of neurons that fall within this range, *Xavier/2* initialisation may be employed, where $\text{var}(w) = 2/n$ [169].

A second approach to ensuring that the input to the activations of each layer is distributed according to a unit Gaussian distribution is to enforce this by means of a so-called *batch normalisation* layer. In this layer, the input is scaled along each feature dimension k such that

$$\hat{x}^k = \frac{x^k - E(x^k)}{\sqrt{\text{var}(x^k)}}, \tag{3.29}$$

resulting in a unit Gaussian distribution for $\hat{x}^k$. The values $E(x^k)$ and $\text{var}(x^k)$ are calculated in respect of the current training batch. Since the batch size during testing may be as small as one, however, a record of the running mean and running variance is kept during training for use in testing scenarios.

The assumption that a layer performs best when the input is distributed in this way may be false in certain instances. In such a case, the normalisation in (3.29) could be harmful to network performance. In order to accommodate such cases, a second transformation is applied, which allows the network to *'unlearn'* the normalisation by adjusting the values of the parameters γ^k and β^k [133]. The new output is then given by

$$y^k = \gamma^k \hat{x}^k + \beta^k.$$

3.5.3 Types of architectures

The *architecture* of a network refers to its overall structure, given by the number of neurons and hidden layers, as well as the manner in which the neurons are connected. This structure is typically determined experimentally during a hyperparameter tuning process [103]. There are, however, various *types* of architectures which share certain properties or structural elements.

The neural network in Figure 3.18 is called a *feedforward neural network* or a *multilayer perceptron*. In such a network, information flows in one direction and there are no feedback connections [103]. Furthermore, the network shown in the figure comprises only *fully connected* layers, in which each neuron of a given layer is connected to every neuron in the subsequent layer, as well as every neuron in the preceding layer [169].

Employing only fully connected layers can result in a significantly large number of weights, rendering the optimisation process difficult and necessitating the acquisition of large amounts of data. If there is some inherent structure in the data, as is the case for time-series data (*e.g.* text or measurements) or images (which consist of two-dimensional arrays of pixel values), *convolutional layers* may be applied instead [103, 169].

Convolutional Neural Networks

A *convolution* is a mathematical operation on two functions of a real-valued argument. The first argument is referred to as the *input* I and the second argument is referred to as the *kernel* or *filter* K that is applied to the input. The output of a convolution is called a *feature map* [103, 207, 169]. In the case where the input is two-dimensional, such as an image, the convolution of I with K is defined as

$$(I * K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n).$$

Convolutions are *commutative*. An equivalent expression of this convolution is, therefore,

$$(I * K)(i, j) = \sum_m \sum_n I(i - m, j - n)K(m, n).$$

The commutative property holds only because the kernel has been *flipped* relative to the input (as the index of the input increases, the input to the kernel decreases). In machine learning applications, however, this property is typically not important. Most libraries therefore implement *cross-correlation* rather than convolution, in which case the kernel is not flipped [103]. The case of cross-correlation is given by

$$(I * K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n).$$

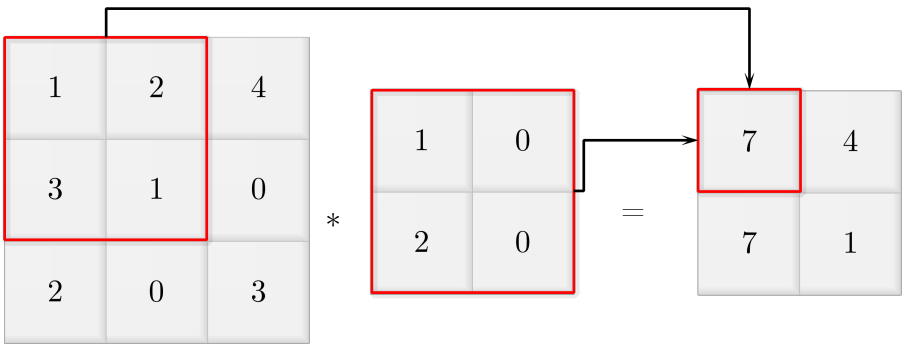


FIGURE 3.26: *Illustration of a convolution.*

For the purposes of this dissertation, the terms cross-correlation and convolution are used interchangeably. The concept of a convolution is best explained by means of an example. Figure 3.26 shows the feature map resulting from the convolution of the filter

$$\begin{bmatrix} 1 & 0 \\ 2 & 0 \end{bmatrix}$$

with the two-dimensional input shown. The convolution is executed by *'sliding'* the filter across the input and computing the dot product of the filter and the region that it covers at every stage. The top left entry of the feature map, for example, is calculated as $1(1) + 2(0) + 3(2) + 1(0) = 7$.

A convolutional layer in a neural network is defined by the width and height of the filter and the number of filters that are applied [169]. The filter depth is always chosen to equal the depth of the input (also referred to as the number of input *channels*). An RGB-image, for example,

has three channels — the input dimensions are $W \times H \times 3$, where W and H are the width and height of the image, respectively. A filter that is applied to this image therefore has dimensions $F \times F \times 3$ (since filters are typically square) and the convolution produces a feature map with one channel. Several different filters may be applied to an input, which then results in a feature map with as many channels as filters. Each filter is then expected to extract a different feature from the input.

As is evident from Figure 3.26, the convolution operation results in an output of smaller dimensions than that of the original input. If several convolutional layers are applied in sequence, the dimensions may shrink too quickly, leading to a possible loss of information. In order to mitigate this effect, *padding* may be applied around the input. This entails adding numbers (typically zeros) around the edges of the input. Adding single zero padding around the input of Figure 3.26, for example, yields

$$I = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 4 & 0 \\ 0 & 3 & 1 & 0 & 0 \\ 0 & 2 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Another motivation for the use of padding is that the corner values of the input would otherwise only be used once, which may again cause a loss of information. If no padding is applied, the convolution is referred to as a *valid convolution*. For a *same convolution*, the level of padding, P, is chosen such that the output size equals the input size [297, 169]. The *stride* S of the filter also has an influence on the output size. This quantity describes the number of elements by which the filter is shifted for each of the output calculations. The stride in Figure 3.26 is, therefore, one. The output dimensions of a convolutional layer are given by

$$\left(\frac{H+2-F}{S}+1\right) \times \left(\frac{W+2-F}{S}+1\right) \times K,$$

where K is the number of filters applied and the other quantities are defined as before. The parameters for such a layer are the $(F \times F \times D) \times K$ weights, where D is the depth of the input and K biases may be added to each of the elements in the feature map [297].

Convolutional neural networks (CNNs) are feedforward neural networks that employ convolution instead of general matrix multiplication (used in fully connected layers) in at least one of their layers [103]. Typically, a non-linearity (such as the ReLU function) is applied to the output of a convolutional layer, before a *pooling* layer is applied. Units consisting of these three layers are repeated until, finally, a fully connected output layer is applied [103, 169].

A *pooling* function replaces the output of a neural network at a certain location with a summary statistic of nearby outputs [103]. *Max pooling* [360], for example, involves selecting the maximum output within a rectangular neighbourhood, whilst average pooling selects the mean of the output values. A pooling layer is defined by the *spatial filter extent* F (the height and width of the square filter), as well as the stride of the filter, and has no parameters [169].

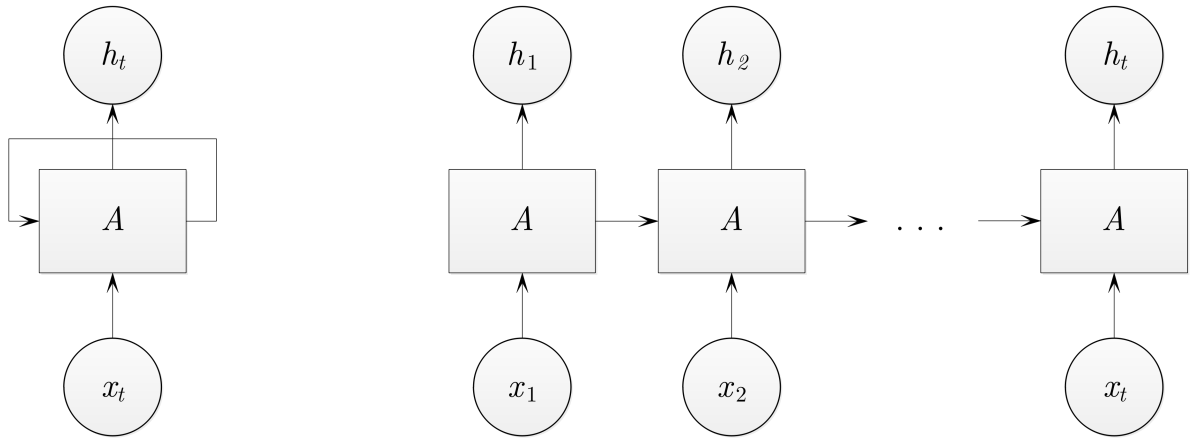
The function of the convolutional layers in a CNN is to *extract* features from the input data. The purpose of the pooling layers, on the other hand, is the *selection* of the most salient features. By incorporating pooling layers, the network becomes robust to insignificant features and *invariant* with respect to local translations in the data [103]. This is one of the advantages of CNNs over networks with only fully connected layers.

Other advantages include the property of *equivariance* to translation and *parameter sharing*. The former means that the output of the network changes in the same way that the input changes.

When processing time series data, a feature that occurs at time step $t = 1$ during training, but at time step $t = 6$ during testing will, for example, yield the same output representation during testing, just shifted by five time steps. Similarly, the network is able to detect an eye in the top left corner of an image, even if all training samples featured eyes in the lower right hand corner of the image [103, 207]. In fully connected layers, the number of parameters that must be stored and trained are equal to the product of the neurons in the preceding layer and the number of neurons in the current layer. Convolutional layers, on the other hand, only require the weights in the filter, which are *shared* for several regions of the input. CNNs are, therefore, more memory-efficient than fully connected networks and typically require less data to reach the same level of performance [103, 169].

Recurrent Neural Networks

Recurrent neural networks (RNNs) are a family of neural networks, first proposed by Rumelhart *et al.* [265]. These networks are designed to process *variable length* sequences of values[17] $x_1, \ldots, x_t$ by using feedback connections. The basic structure of an RNN is shown in Figure 3.27(a). Inputs x_t are passed through an arbitrary neural network A one after the other. At each time step, an output h_t is produced, which is also passed back to the network as a secondary input during the next time step. This recursion is *'unfolded'* in Figure 3.27(b).

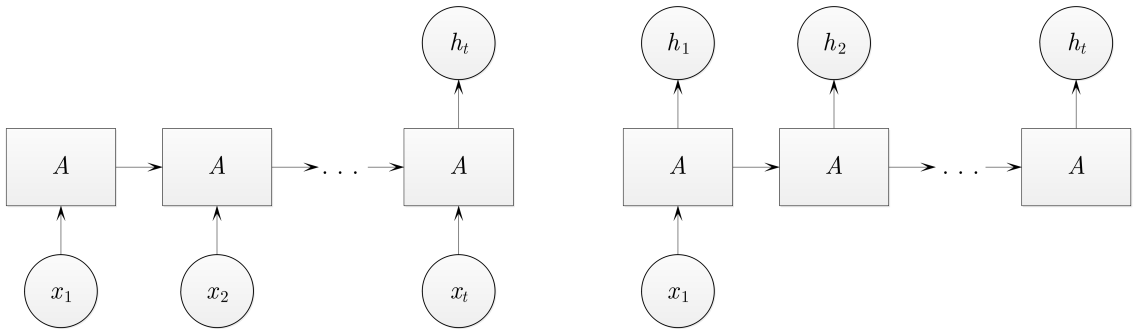


(a) *An RNN* (b) *'Unfolded' representation of the same network*

FIGURE 3.27: *Basic structure of a recurrent neural network in (a) standard and (b) 'unfolded' notation. The sequential inputs $x_1, \ldots, x_t$ are passed through some neural network A in the same sequence, producing the output values $h_1, \ldots, h_t$, which also serve as feedback to the network [219].*

Feedforward neural networks are typically used to model *one-to-one* relationships, such as the relationship between an image and its class label. RNNs can be used to model various other types of relationships. Adopting the structure in Figure 3.27, for instance, *many-to-many* relationships can be modelled, as is useful for machine translation or event classification. Others include *one-to-many* relationships, occurring in image captioning tasks, and *many-to-one* relationships, which are useful for language recognition or text classification tasks. These structures can also be modelled by means of RNNs, as shown in Figure 3.28.

[17] The *values* $x_1, \ldots, x_t$ referred to here are of the variable type *tensor*. Tensors are generalisations of scalars (that have no indices), vectors (that have exactly one index) and matrices (that have exactly two indices) to an arbitrary number of indices [263].



(a) *Many-to-one* (b) *One-to-many*

FIGURE 3.28: *Relationships that can be modelled by an RNN model, in addition to the many-to-many relationship shown in Figure 3.27 [169].*

The *hidden state* of a neural network may be expressed at each time step as $A_t = \theta_c A_{t-1} + \theta_x x_t$ and the observed output as $h_t = \theta_h A_t$, where θ_c, θ_x and θ_h, are the parameters to be learnt for the *cell* (the neural network, A), the input x and the output h, respectively. Unlike feedforward neural networks, these parameters are shared by the network during every time step, thereby greatly reducing the number of parameter values to be optimised during training [358].

As extensively discussed by Bengio *et al.* [25], the central problem for optimising RNNs is the modelling of *long-term dependencies*. Consider the example of a long text document, which contains the phrase "I grew up in Italy" at the beginning of the document, and the phrase "I speak fluent Italian" towards the end of the document. If a network is to predict, for example, the last word of the second phrase, it should consider the information contained at the beginning of the document. The gap between the phrases may, however, be significantly large [219]. In order to model this long-term dependency, the network must compute the quantity $\theta_c^t A_0$. During optimisation by gradient descent, the problems of vanishing and exploding gradients, therefore, become even more prominent than in feedforward neural networks.

It can be shown that the problem of vanishing gradients exists when the eigenvalues of θ_c are smaller than one. Exploding gradients, on the other hand, occur when the eigenvalues are greater than one. The latter problem may be solved by introducing *gradient clipping*, which entails setting the gradient to a predefined maximum value in the case that it should overshoot this value in any given iteration [235]. The problem of vanishing gradients, on the other hand, can originate either from the weights themselves or from the activation functions in the network, which are typically *tanh* functions for RNNs [169]. Model parameters may be chosen such that the eigenvalues are exactly equal to one in order to combat the first source.

A successful strategy toward overcoming the problem originating from the second source is adopting an alternative network architecture, as proposed by Hochreiter and Schmidhuber [124]. In this *long short-term memory* (LSTM) network, the neural network A, which consists of a single layer neural network with a *tanh* activation function for a standard RNN, is replaced by a more complicated structure, as shown in Figure 3.29.

The key idea behind an LSTM network is the *cell state* C, which flows horizontally along the top of the network and is subject to only a few, linear transformations. It is also possible for the information to flow through the unit unchanged along this path. Intuitively, this provides a *'pathway'* for the gradient to continue to flow, even if the gradients of the *tanh* function in the network are *vanishing* [169].

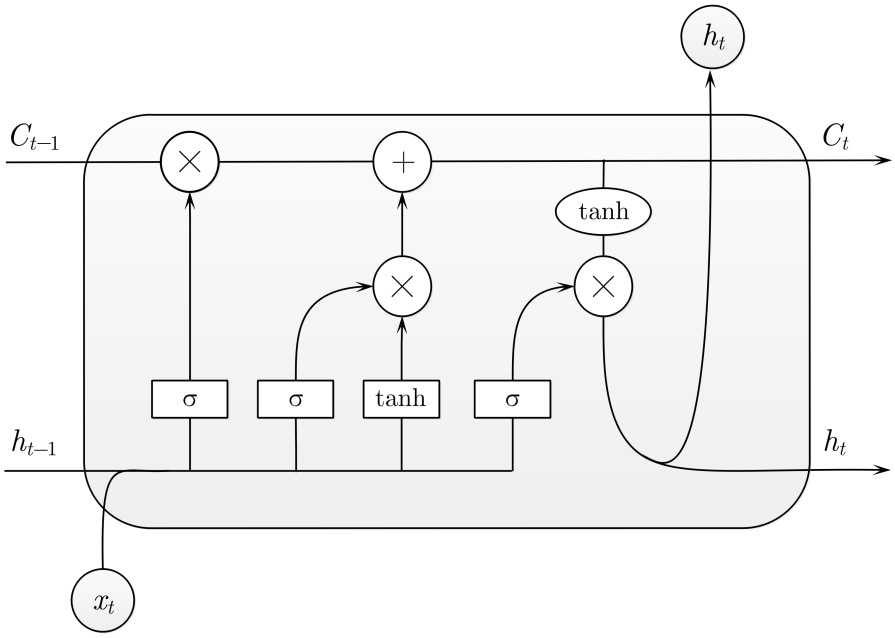


FIGURE 3.29: *An illustration of the LSTM network architecture [219].*

The cell state can be altered by introducing the so-called *gates* into the network. The first is the *forget gate*, where the old cell state is multiplied by the output of a sigmoid layer, whose value is between zero and one, and is a function of the input x_t and the output from the previous time step h_{t-1}. A value of zero indicates that the previous cell state should be forgotten, whilst a value of one suggests that the prior information is entirely important. The second gate in the network governs the addition of new information to the cell state. The values of x_t and h_{t-1} are passed through a *tanh* function, as in a standard RNN. In an LSTM network, however, the output of this activation is then multiplied by the *input gate*, given by another sigmoid layer, which chooses how much (as a value between zero and one) of this output should be added to the cell state. The last gate is the *output gate*, which decides which values of the cell state should be returned. It functions similarly to the input gate, except that the new cell state C_t is passed through the *tanh* and sigmoid functions instead of x_t and h_{t-1} [219].

A popular variation of the LSTM network is the *Gated Recurrent Unit* (GRU), which combines the forget and input gates into a single *"update gate,"* among other small changes, resulting in a slightly simpler model [219, 358].

Another new development in the realm of recurrent neural networks is the *attention mechanism*. It is inspired by the human visual system, which is able to focus on a specific region of an image, whilst blurring out the remainder of the image, and adjusting this focal point over time. When using the attention mechanism, each output is based on a weighted combination of the input states, as opposed to just the previous time state, allowing the network to select specific input values that are relevant to the output during the time step in question (by assigning large weights to these inputs) [358]. Bahdanau *et al.* [13] applied this concept to machine translation tasks, allowing the network to select words from the source sentence that are relevant to the next word to be predicted in the target sentence, based on what has been translated already.

Autoencoders

It was mentioned earlier in this chapter that one of the significant advantages of neural networks is their ability to automatically extract features from input data, thereby allowing the laborious task of feature engineering to be transferred to the model itself. It stands to reason, then, that neural networks can be used to create lower-dimensional representations of data.

An *autoencoder* is a specific type of feedforward neural network which does exactly that. It consists of three layers, namely an input layer, a hidden layer and an output layer, as shown in Figure 3.30. The target values, in this case, are set equal to the input values. The network therefore finds a hidden representation of the input values in the hidden layer using an *encoder function*, and subsequently attempts to recreate the input from this hidden representation using a *decoder function*. The network is trained to minimise some reconstruction error $L_i(x_i, \hat{x}_i)$. The activation of the hidden layer is then taken as the learnt representation of the input. Due to the non-linear nature of the activation functions in the neural network, it is more powerful than its linear counterparts, namely PCA and LSA, described in §2.8 and §2.3.2, respectively [358].

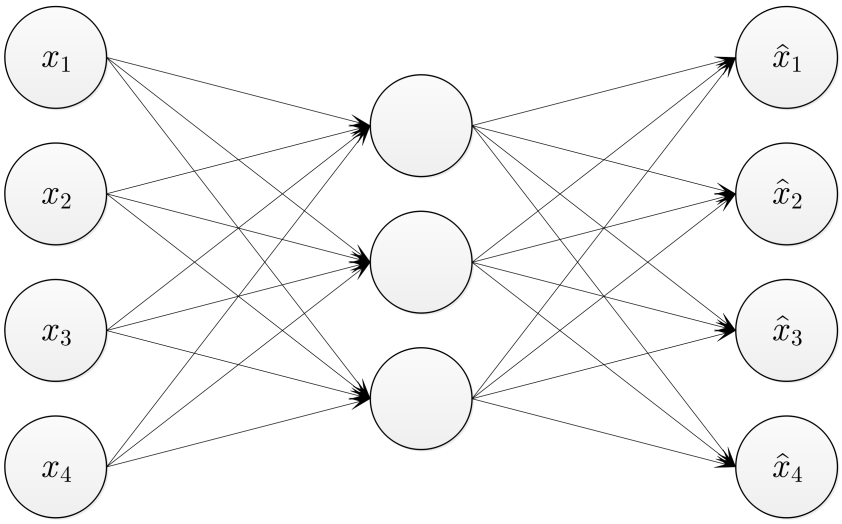


FIGURE 3.30: *An autoencoder architecture for an input vector* $\boldsymbol{x} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}$.

A simple variation on the network shown in Figure 3.30 is the *denoising autoencoder* (DAE) [326], in which *noise* (typically sampled from a Gaussian distribution) is added to the input and the network is trained to *denoise* this input. The objective of this extension is to create representations that are more *robust* to small changes in the input data [358].

Autoencoders are often *stacked* into several layers in order to describe multiple levels of representations of the input [358]. A popular such architecture is the *Stacked Denoising Autoencoder* [327].

In contrast to the other algorithms described in this chapter, autoencoders function in the *unsupervised* learning paradigm. Their parameters can be used to initialise supervised neural networks, which has been shown to improve performance over random parameter initialisation [358]. Other approaches add a supervised component to the autoencoder in order to train the unsupervised *representation learning* and supervised classification tasks simultaneously. Rasmus *et al.* [254], for example, created *Ladder Networks* by introducing lateral connections into a denoising autoencoder structure and trained this network to minimise an unsupervised reconstruction loss and a supervised classification loss simultaneously.

3.6 Chapter summary

This chapter contained an introduction to the notion of machine learning, as well as various types of learning that prevail in this field, namely supervised, unsupervised, semi-supervised, weakly supervised and reinforcement learning. Furthermore, the differences between classification and regression problems, generative and discriminative models, as well as probabilistic and scoring classifiers was illustrated.

Subsequently, a general procedure for training and evaluating machine learning models was outlined, and this was followed by a more detailed description of the process of hyperparameter tuning, model generalisability and the bias-variance trade-off, as well as several performance evaluation metrics, including accuracy, precision, recall, the F-measure and the AUC value.

Several machine learning algorithms were described next, namely kNN, tree-based methods, SVMs, naïve Bayes classifiers, binary and multi-class logistic regression (maximum entropy). This was followed by a discussion on ensemble learning, including the rationale behind this methodology and possible reasons for its superior performace, as well as considerations for constructing ensembles. Detailed descriptions of three popular ensemble learning methods, namely bagging, boosting and stacking, were also provided, and this was followed by a brief review of ensemble pruning methods. Finally, a review of deep neural networks was provided, including training and design considerations specific to this type of machine learning algorithm, as well as an account of several types of neural network architectures, namely feedforward, convolutional and recurrent neural networks, as well as autoencoders.

CHAPTER 4

Sentiment analysis

Contents

The purpose of this chapter is to review the literature pertinent to this study. First, the typical process followed to analyse data of an unstructured nature is described. Subsequently, several techniques for processing such data are introduced, including tokenisation and normalisation, as well as vectorisation. The field of sentiment analysis is then reviewed, including a brief outline of the history of the field, the various tasks and levels of analysis that exist within it, as well as the techniques that are commonly used to analyse and synthesise sentiment.

4.1 Analysing unstructured data

Data are defined, according to the Oxford dictionary, as "facts and statistics collected together for reference or analysis" [228]. Although they are typically of little value or interest for decision makers in their *raw*, unprocessed form, they may be *analysed* in order to derive valuable information and insights regarding the process or system which underlies them.

The applicability of various analysis techniques depends largely on the nature of the data. Collections of raw data may be categorised, for example, by the *form* in which they are presented. Jiang *et al.* [139] differentiate between four data forms in the realm of data analytics or *data mining*[1]. The first are *textual* data forms, which comprise an extensive collection of characters that are not arranged according to any definitive structure or format. *Time-series* (time-varying)

[1]Data mining is concerned with the recognition of patterns in data for the purpose of *knowledge discovery* [32, 216, 334].

data are typically stored using *temporal* data forms, where each entry contains information linking it to a specific period of time. *Transactional* data forms also contain a time dimension, but are distinguished from temporal data forms by the addition of a reference to one or more objects associated with a particular event, such as the items involved in a customer purchase. Lastly, data that are organised in tables constitute *relational* data forms. Each row in such a table represents a *record* and each column represents an *attribute*, or *property*, of that record [144].

In broader terms, data may be classified as either *structured* or *unstructured* [334]. The relational and textual data forms in the taxonomy of Jiang *et al.*[139] are typical examples of these respective categories. In practice, a structured data set may be encountered in the form of a spreadsheet containing the personal details of active customers. Each record in the table in this case represents a customer, whilst the attributes contain information about the customers, such as their names and contact numbers, or the organisations with which they are affiliated. Unstructured data may present itself in the form of e-mails, written reports or customer complaints in SMS form. This type of data is not limited to the written medium, but may also include visual and audio data such as recorded telephone conversations, images or video footage. The analysis of each of these media is a distinct field of study [350]. In this dissertation, the focus is on the analysis of natural language, specifically in written text format.

Traditional methods of data analysis are often tailored to data that are presented in a structured form [73]. The analysis of unstructured data, however, may perhaps be of even greater importance, since a considerable proportion of business-relevant information is of an unstructured nature. Many estimate this proportion to be as large as 80–90% [73, 107]. The extraction of knowledge and insights from data in various (structured and unstructured) forms is referred to as *data science* [73]. According to O'Neill and Schutt [222], most data science projects follow the generic process shown in Figure 4.1.

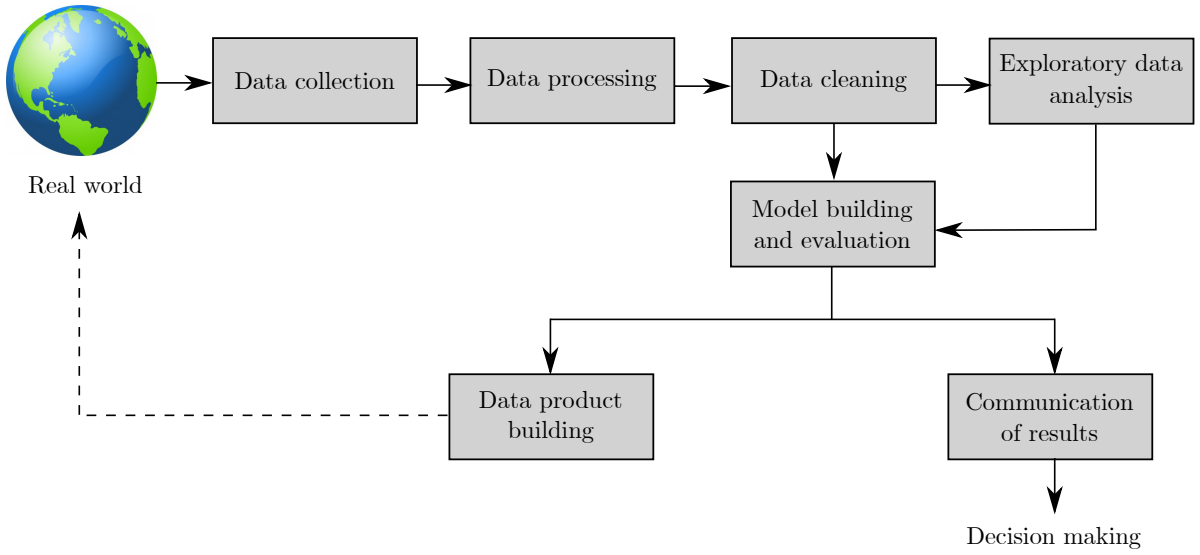


FIGURE 4.1: *The data science process (adapted from O'Neill and Schutt [222]).*

In the first step, data generated by real-world processes or systems are *collected*. Since most analysis techniques (especially computational approaches) rely on a structured representation of their input data, an important next step towards deriving insights from unstructured data is

its transformation into a structured, often numerical, representation [334]. This is referred to as *data processing*.

The objective of the *data cleaning* step is the detection and removal of errors and inconsistencies present in the data. According to Nisbet *et al.* [216, p. 40], this encompasses three principal activities: Imputation (the filling of blank entries), the treatment of error values, and the management of outliers.

The clean, structured data may now be used for *model building and evaluation*. It is common and useful, however, to precede this step with an *Exploratory Data Analysis* (EDA). This entails generating summaries, such as the minimum, maximum and mean values, of the variables in the data set and visualising the relationships between sets of variables. The underlying philosophy of EDA is that an understanding and intuition of the data may be gained in this way, leading to improvements during model development [222]. This step may also uncover inconsistencies that were missed during data cleaning and elicit the need for further data collection.

Subsequently, relevant models are built and appropriate analysis techniques are applied to the data to solve the problem in question. The model results may then be interpreted, visualised and *communicated* to stakeholders in order to aid in decision making. Alternatively, these models may be used to develop a *data product*, such as an algorithm, tool or DSS, that users in the real world interact with and employ as an aid in repeated decision making [222].

The remainder of this section contains a discussion on common data processing operations pertinent to the problem of sentiment analysis. Relevant approaches in the model building steps of Figure 4.1 are addressed in §4.2. Finally, techniques pertaining to data product building are addressed in §4.2.4 and Chapter 5.

4.1.1 Tokenisation and normalisation

The objective of the data processing step in Figure 4.1 is, in essence, to transform unstructured natural language into a format suitable for analysis by a computer. This concept is best illustrated by means of a simple example. Consider the documents shown in Table 4.1.

Document	Contents
A	Scary movies are great!
B	Our children are scared.

TABLE 4.1: *Simple example of a corpus with two textual documents representing unstructured data.*

Each document in the table consists of an unstructured string of text, containing a differing number of characters. For a human assessor, the analysis of these sentences is trivial. One may say, for example, that the first sentence expresses a positive sentiment towards films of the horror genre, and that the second statement is more objective, its sentiment bearing a negative tendency. A computer, however, merely sees two *strings* of characters — in order to facilitate an analysis, the text must first be translated into a language it can comprehend.

An alternative, structured representation of the data in Table 4.1 is shown in Table 4.2. In this representation, which is perhaps the simplest, the text is first segmented on the spaces between words. Each of the documents is then represented by a series of individual *tokens* rather than one long string. Document A, for example, becomes $\mathcal{A} = \{\text{“}Scary\text{”}, \text{“}movies\text{”}, \text{“}are\text{”}, \text{“}great!\text{”}\}$. This segmentation process is referred to as *tokenisation* [132]. The representation of a text as a

set of its words, without regard for possible dependencies between words, is commonly referred to as the *bag-of-words* model [172, 243].

Document	Scary	movies	are	great!	Our	children	scared.
$\mathcal{A}$	1	1	1	1	0	0	0
$\mathcal{B}$	0	0	1	0	1	1	1

TABLE 4.2: *A structured representation of the corpus in Table 4.1. The entries in the table are binary variables taking the value 1 if the word at the column head is contained in the document at the row head, or 0 otherwise.*

Subsequently, a *dictionary* of all unique tokens, referred to as *types*, that are contained in the set of documents $\mathcal{C} = \{A, B\}$ is generated [333, 334]. This set, $\mathcal{C}$, is commonly referred to as a *corpus* [216, p. 174]. In this case, the dictionary is the set $\mathcal{D} = \{$"*Scary*", "*movies*", "*are*", "*great*!", "*Our*", "*children*", "*scared.*"$\}$. A table may then be constructed, in which each document in the corpus constitutes a row, and the columns, or *features*, are the token types contained in the dictionary. Each table entry, a_{ij}, in this case, represents the presence or absence of a token of type j in document i. The unstructured data have thus been transformed into a relational data form and a computer may now employ, for example, traditional matrix operations in order to compare the documents and draw conclusions from the data. Hereafter, a document is denoted as vector, $\boldsymbol{d}$, and a corpus is denoted as a matrix, $\boldsymbol{C}$.

Upon closer examination of Table 4.2, it is evident that the tokenisation procedure employed in this example is too simplistic. It may, for example, be undesirable to leave punctuation marks attached to the tokens. If a new document contained the word "great" followed by anything other than an exclamation mark, this would be seen as a token of a different type to that of the fifth column in the table. On the other hand, for contracted words such as "isn't", or abbreviations such as "U.S.A", the punctuation should indeed be seen as a part of the token. According to Weiss *et al.* [333], spaces and any characters from the set $\{(,), <, >, !, ?, "\}$ are always delimiters of tokens, whereas characters from the set $\{., ,, :, -, '\}$ may or may not serve this function, depending on their environment. It is therefore suggested that existing tokenisers be adapted to their specific context. Furthermore, it is noted that tokenisation is a language-dependent process [333].

Another issue which emerges from Table 4.2 is the fact that the tokens "Scary" and "scared" are seen to be of different types, although they should, ideally, be grouped in a base form common to both words, such as "scare." Converting tokens to a standard form is referred to as *normalisation*. This includes many subprocesses, such as the conversion of text to all lower-case, the correction of spelling errors and shorthand, as well as *stemming* and *lemmatisation* [243]. The objective of both stemming and lemmatisation is to reduce inflectional or derived forms of a word to a common base form. Whilst stemming is typically a crude process which removes word endings according to some heuristic, lemmatisation makes use of a vocabulary and a morphological analysis[2] of words to return the base form of a word, known as the *lemma* [185].

The most widely used stemming algorithm for the English language is *Porter's Stemming Algorithm*, which has repeatedly been shown to be empirically effective since its publication in 1980 [32, 185]. It consists of five sequential phases of word reductions. The first phase is concerned with plural and past participle forms. The word endings "sses", "ies" and "ss," for example, are reduced to "ss", "i" and "ss", respectively, whilst a trailing "s" is removed from words ending in this letter. During the second phase, words containing a *double suffix* are re-

[2] In linguistics, morphology is the study of words, their formation and internal structure [9].

duced to a single suffix representation. "Hopefulness," for example, is reduced to "hopeful." The third and fourth phases follow a similar approach, iteratively altering and removing word endings of certain patterns until all suffixes have been removed. During the fifth phase, final transformations are performed, removing an "e" or a second "l" at the end of certain words and returning the stemmed form of the original word as output. Each transformation considers the length of a word, as well as the vowel-consonant patterns contained within it, so as to avoid *overstemming* [110, 324]. Other stemming algorithms also exist, including the *One-pass Lovins Stemming Algorithm*, the *Paice/Husk Stemming Algorithm*, and the *Lancaster Stemming Algorithm* [29, 185].

In some cases, the stem returned by a stemming algorithm does not correspond to the lemma of the word contained in the dictionary. This can often be rectified by a process of lemmatisation. Manning *et al.* [185] noted, however, that the resulting form of the stemming process is irrelevant, as long as it is equivalent for other forms of input. Furthermore, it has been demonstrated that the increase in performance when using a lemmatiser over a stemming algorithm is typically modest [185].

4.1.2 Vectorisation

Each row in Table 4.2 is the result of a data *vectorisation* process [243]. In this case, each document was represented by a feature vector with binary elements taking the value 1 if the corresponding term in the dictionary is present in the document, or 0 otherwise. This text representation is referred to as the *Bernoulli document model* [28]. There are, however, several other possible sets of features that may be used in order to represent text in a vectorised format. In this section, common types of features are described, focusing on the literature relevant to sentiment analysis and general findings within this field.

Term-based features

Instead of employing the *presence-based* Bernoulli model, it is common in natural language processing applications to represent a text by means of a *frequency-based* feature vector [350]. In the *multinomial document model*, for example, each entry of the feature vector is an integer value corresponding to the frequency count of that term in the document [28]. Alternatively, a frequency weight may be used, which takes into account the frequency of a term in a document relative to the frequency of the same term in the entire corpus.

One popular such frequency weight is the *term frequency-inverse document frequency* (TF-IDF) weight [231, 243]. Many variants of this weighting scheme exist, but it is typically composed of two terms: The *term frequency* (TF) and the *inverse document frequency* (IDF). The former is calculated as

$$T(t, \boldsymbol{d}) = \frac{f_{t,\boldsymbol{d}}}{\sum\limits_{t' \in \boldsymbol{d}} f_{t',\boldsymbol{d}}},$$

where $f_{t,\boldsymbol{d}}$ is the frequency count of term t in document $\boldsymbol{d}$ and the denominator represents the total number of terms in the document. By normalising the frequency count over the length of the document, a bias toward longer documents, in which frequency counts of a given term tend to be higher, is avoided. The inverse document frequency is a measure of the information provided by a term. The assumption is that terms which occur frequently across documents, such as "the," "are" and "is," provide little meaningful information and should receive a lesser

weight than terms which occur more rarely. It is calculated as

$$I(t, \boldsymbol{C}) = \log \frac{N}{n_t},$$

where N is the total number of documents in the corpus $\boldsymbol{C}$ and n_t is the number of documents in the corpus which contain the term t. The value is therefore equal to zero if the term is contained in all documents of the corpus, and increases with an increase in the rarity of the term. The frequency weight $F(t, \boldsymbol{d})$ is finally calculated as the product of $T(t, \boldsymbol{d})$ and $I(t, \boldsymbol{C})$ [243].

In topic classification problems, models using frequency-based feature vectors typically outperform those which employ presence-based features [231, 232]. In direct opposition to this observation, however, Pang *et al.* [232] found a significant increase in performance with the use of presence-based features in sentiment polarity classification problems. In light of this, it is suggested that, while the repeated occurrence of certain keywords may highlight a specific topic, the same is not true for the overall sentiment of a text [231].

Some researchers argue that the position of a token within a document can influence its effect on the overall sentiment of the document [231]. It has, for example, been claimed that the last few sentences of a product review often best summarise the document's overall sentiment [230]. Positional information, such as whether the term appears at the beginning, middle or end of a document, is therefore often encoded into feature vectors [157].

In order to capture a wider context, one may also use higher order *n-grams*, rather than individual terms (*unigrams*) as document features. These features represent groups of adjacent words in their original word order [350]. It stands to reason that a better performance for classifying sentiment can be achieved by viewing, for instance, negated phrases such as "not happy" as a single attribute. There is, however, controversy in the literature in this regard. Whilst Pang *et al.* [232] found that unigrams outperformed bigrams in classifying the sentiment polarity of movie reviews, Dave *et al.* [63] reported bigrams and trigrams to yield better results in product review classification [231]. It has, in fact, often been found that the problem of sentiment classification, and therefore also the effectiveness of certain feature sets, is domain-specific and context-sensitive [52].

Linguistic features

Term-based features are often referred to as *surface features*, along with other superficial attributes such as the mean word count or sentence length of a text. *Linguistic features*, on the other hand, require deeper analysis of the text and its structure. Such features include *parts-of-speech* (POS) tags of individual tokens, as well as the syntactic relations between words, which may be extracted using *parse trees* [93].

POS tagging entails attaching a tag to each token in a document, indicating the term's part of speech. For example, the sentence "the man ran away" becomes {("the", "DT"), ("man", "NN"), ("ran", "VBD"), ("away", "RB")} after a tokeniser and a POS tagger have been applied. The tags applied in this case are explained in Table 4.3.

POS tags are popular in many text analysis problems. One major reason for this is that POS tagging may serve as a crude form of word sense disambiguation [231]. The word *great* in the phrase "the food was great" is indicative of a positive sentiment. The same word, however, does not bear any sentiment in the statement "I am from Great Britian." Identifying the word as an adjective in the former context and as part of a proper noun in the latter context can successfully improve the accuracy of a model.

POS tag	Meaning
DT	Determiner
NN	Noun, common, singular or mass
VBD	Verb, past tense
RB	Adverb

TABLE 4.3: *Examples of POS tags in the* `Python` *Natural Language Toolkit (*`NLTK`*) and their meaning [29].*

It has, furthermore, been found that there is a high correlation between the presence of adjectives and the subjectivity of a sentence [118], leading many researchers to focus on the use of adjectives as features in sentiment classification [231]. In another approach, Turney [321] proposed using phrases which correspond to pre-specified parts-of-speech patterns as features, most of which contain adjectives or adverbs. Results by Pang *et al.* [232], however, indicate that using adjectives alone results in inferior performance compared to the use of the most frequently occurring unigrams. Although adjectives have been found to contribute significantly to the level of subjectivity of a text, it may, therefore, be concluded that other parts of speech can be equally good indicators of sentiment [231].

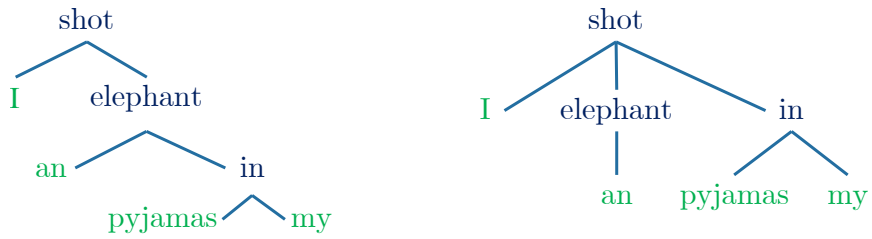
Instead of merely extracting lexical information provided by POS tagging, some researchers have also attempted to incorporate syntactical dependencies of a text into feature sets by *parsing* the text in order to generate *dependency trees* [231, 306].

To *parse* a sentence is to resolve it into its components and to describe their syntactic roles [229]. This structure is commonly represented by trees, which are constructed according to either the *constituency model* or the *dependency model* [2]. The former approach is concerned with the way in which sequences of words *combine* to form constituents. A sentence is typically composed of a *noun phrase* and a *verb phrase*, each of which may be further decomposed. The noun phrase "The dog", for example, is composed of the determiner "the" and the noun "dog." Dependency-based parse trees, on the other hand, are commonly used in the realm of sentiment analysis and focus on the *relationship* between words in the sentence [29, 231].

Parse trees are often used to recognise and illustrate ambiguity in natural language. Consider the following excerpt from the 1930 film *Animal Crackers* starring Groucho Marx [119]: "One morning I shot an elephant in my pajamas. How he got into my pyjamas, I don't know." Parsing the ambiguous part of the text yields two dependency trees, which indicate the two possible interpretations of the sentence, as shown in Figure 4.2. Specifically, in Figure 4.2(a), the pyjamas are directly related to the elephant (which appears to be inside them), while in Figure 4.2(b), they are related to the verb *shot*, indicating that the shooting was executed whilst the subject was wearing pyjamas.

Whether including such dependency information in features results in an improved performance in sentiment classification problems is a matter of debate in the literature. Dave *et al.* [63], for example, found no performance increase with the use of such features, whilst Gamon [93] concluded that the use of features stemming from an abstract linguistic analysis consistently contributed to the accuracy of sentiment classification.

Parsing and POS tagging can also be used as a basis for modelling *valence shifters* or *sentiment shifters*, such as negation, diminishers and intensifiers. These are expressions that can alter a sentiment orientation from positive to negative, or *vice versa* [176, 231]. The earliest work addressing negation is by Das and Chen [62], who attempted to model negation directly by attaching "-NOT" to words occurring close to negation terms [231]. Whilst this works for many constructs, it incorrectly detects negation in sentence structures such as "not only... but



(a) *First interpretation* (b) *Second interpretation*

FIGURE 4.2: *The two dependency trees generated when parsing the ambiguous sentence "I shot an elephant in my pyjamas" (adapted from Bird and Ewan [29]).*

also" [176]. Furthermore, negation may be expressed in more subtle ways, as in the phrase "the movie avoids clichés" [231]. In order to address these shortfalls, Na *et al.* [209] proposed reversing the polarity of phrases which exhibit certain parts-of-speech patterns. They reported a 3% increase in accuracy, and suggested that a deeper syntactic analysis may lead to a further improvement in accuracy.

Topic-oriented features

The sentiment polarity of a text depends on the topic addressed in the text [172, 231, 350]. The same word can have a different semantic orientation in different contexts. The word "complex" may, for instance, be viewed as positive in a movie review where reference is made to a "complex plot." The same word is, however, considered to carry a negative weight in a product review stating that a camera is "complex and difficult to use" [172]. In spite of this, topic-oriented features are not as common as term-based and linguistic features, and are often not explicitly mentioned in surveys on sentiment analysis [176, 190].

Mullen and Collier [205] examined the effectiveness of features based on topic information by (manually) adding tags to sentences which indicated whether they refer to a specific topic, or whether they precede or follow such sentences. Their results suggest that incorporating topic information into existing models is beneficial for performance. Wilson *et al.* [340] added the *document topic* as a feature directly in their attempt to recognise contextual polarity in phrase-level sentiment analysis. A list of 15 topics was composed prior to the analysis, including both specific topics, such as the '2002 presidential election in Zimbabwe', and general topics, such as 'economics.' Each phrase in the corpus was then assigned to a topic in one of the feature columns.

Topic-sentiment interactions can also be modelled using dependency trees, which can be employed to identify the subject of an opinionated phrase [231]. In the dependency tree shown in Figure 4.2, for instance, *"elephant"* is identified as the object of the sentence. Objects of opinionated phrases may be identified in a similar manner. Choi and Kim [52] used syntactic dependency to identify *sentiment topics* of opinionated sentences, and used this information to build a lexicon of *sentiment clues* whose polarity values are contextually driven (such as the word "complex" in the example above).

Other approaches model topic and sentiment jointly, rather than introducing topic-related features. Li *et al.* [172], for example, introduced a sentiment layer to the popular topic model LDA, as is explained in the following section.

4.1.3 Dimensionality reduction and feature selection

When tokens contained in the dictionary are used as features to represent text, it may appear as if the feature space becomes excessively or impracticably large. Often, this concern is unwarranted, since most documents only use a small subset of the words contained in the dictionary, resulting in a *sparse* matrix (in which most elements are zero). Models using the feature space may therefore leverage this property by storing only positive values [243, 334].

On the other hand, these sparse vector representations have significant disadvantages. First, statistical models are more difficult to train with such data. These models are then prone to *overfitting* and require larger sample sizes in order to train effectively [32, 309]. Secondly, representing text data in this way provides no meaningful information to the model as to the relationships that may exist between words, as illustrated in Figure 2.9.

In the remainder of this section, methods of dimensionality reduction are introduced, some of which also address the problem of data sparsity.

Reducing the size of the dictionary

One method for dimensionality reduction has already been mentioned, namely stemming or lemmatisation. By grouping several inflectional forms of words into one feature, dimensionality is reduced. Similarly, one may remove words that contribute little valuable information, such as the common words "and," "or" and "the." These words are called *stopwords* and are often removed from the feature space [32, 333].

After stopwords, the most frequent words in a corpus tend to be important words and are typically retained, whilst rarer words are often considered to be misspellings and deleted [333]. In the *frequency pruning* approach, attention is drawn to the fact that terms in a text are distributed according to a power law distribution[3]. Dimensionality is reduced, in this case, by removing terms that occur in fewer than n documents, where n is typically chosen as 5, or more than m times, where m is a percentage of the number of documents [32]. As was mentioned previously, however, infrequent terms can have as large an impact on the sentiment polarity of a text as frequent terms (whilst this is not true for the topic of a text). The use of frequency pruning in sentiment analysis, therefore, remains questionable.

Feature transformation

Another approach toward dimensionality reduction of the word feature space involves the use of *feature transformation methods* such as PCA. As was mentioned in the detailed description of this method in §2.3.1, it is a useful tool for summarising data in fewer dimensions, while minimising the loss of information. On the other hand, a drawback of this method presents itself in a possible loss of interpretability of the data when the original features are replaced by their linear combinations [243]. In the realm of natural language processing and sentiment analysis, however, interpretability may be retained by viewing the new dimensions as clusters of similar or associated concepts conveyed in the text. Poria *et al.* [242], for example, used PCA to map features of *concepts*[4] onto a so-called *affective space,* in which concepts that convey the same emotion tend to appear close to each other. The concepts *beautiful day* and *birthday party*,

[3] The power law distribution is the mathematical basis for the *Pareto principle* or the '80-20 rule'. In the case of term distribution in text, the top 20% of most frequent terms in a corpus occur significantly more often than other terms in the corpus [31].

[4] *Concepts* refer here to chunks of sentences separated by a parsing algorithm [242].

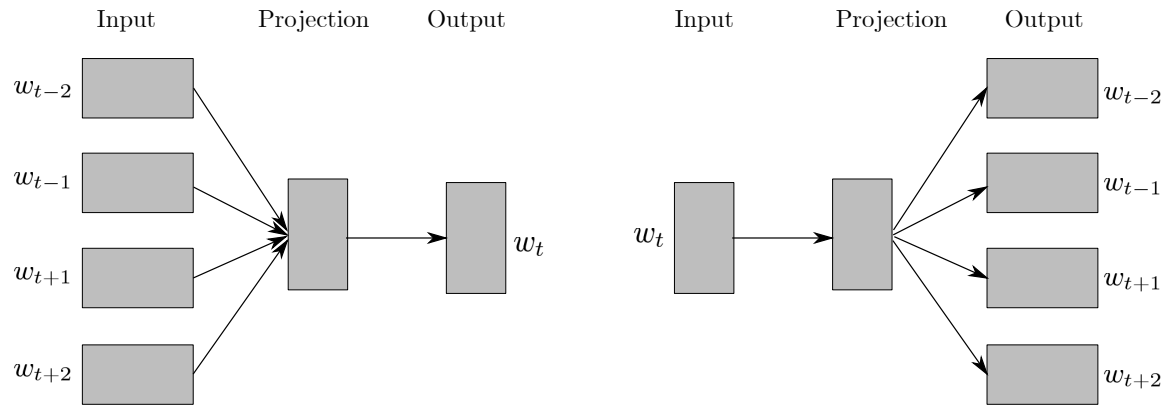
for instance, are close to each other in the transformed vector space, whilst *feel guilty* and *shed tear* are shown to have completely different directions.

In fact, this is the idea behind the closely related method of LSA, described in §2.3.2. The concept is based on the *distributional hypothesis*, which states that words appearing in the same contexts typically convey similar meanings. LSA is considered a *vector space model*, since it represents, or *embeds*, words in such an affective vector space. Furthermore, it is an example of a *count-based* method due to the fact that it employs the statistics of the co-occurrence of words to create this vector space. *Predicive vector space models* also exist, such as those used in *word2vec*, which is covered in the next section [309].

Representation learning

In recent years, *machine learning* and, particularly, *deep learning* techniques have become increasingly popular. In line with this influence, an alternative to creating dense vector representations through linear transformations of the sparse vector space involves the use of algorithms which can *learn* useful data representations [358].

A popular model for producing dense vector representations of individual words, or *word embeddings*, is Google's *"word2vec"* [104]. It has two variants, namely the *continuous bag-of-words* (CBOW) model and the *Skip-Gram model*, both of which make use of neural networks that are trained to learn word vector representations, which improves performance for a given prediction task [195]. Schematic representations of these models are shown in Figure 4.3. The CBOW model is trained to predict a target word w_t, given its surrounding context words. Given the phrase *"the brown horse jumps,"* for example, the model may try to predict the word *"jumps,"* given the remaining words as input. As is shown in Figure 4.3(a), the input vectors are averaged, rendering the word order irrelevant (hence the reference to bag-of-words model) [194]. The Skip-Gram model, on the other hand, attempts to predict a selected number of words in the vicinity of the current word [194, 309]. For the same example, it may then be given the word *"jumps"* as input and attempt to predict the words *"brown"* and *"horse."* Treating the context as a single observation in CBOW has a smoothing effect, resulting in better performance for small data sets. For large data sets, Skip-Gram models, which treat each pair of input and target context word separately, appear to work better [309].



(a) *The CBOW model* (b) *The Skip-Gram model*

FIGURE 4.3: *Schematic representation of the CBOW and Skip-Gram models (Mikolov et al. [194]).*

The resulting learnt word embeddings have been shown to effectively encapsulate semantic relationships between words in the vector space. A famous example illustrates that vectors produced by a Skip-Gram model exhibited arithmetic relationships that make sense semantically. More specifically, the vector operation $vector(\text{“}King\text{”}) - vector(\text{“}Man\text{”}) + vector(\text{“}Woman\text{”})$ resulted in a vector that was closest to the vector representation of the word *Queen* [194].

The word embeddings of terms contained in a document can be used as features for a non-neural classifier or as an input to a second neural network, which can, in turn, be trained to produce a dense vector representation of that document. Alternatively, this dense document vector can be learnt directly from the bag-of-words representation of the text. In cases where neural networks are used solely for the purpose of representation learning, auto-encoder architectures are a popular choice [358].

4.2 Fundamentals of sentiment analysis

The field of *sentiment analysis* refers broadly to the study of people's disposition towards certain targets, typically including products or services, public figures, events or current issues. By analysing observations of people's actions in the form of facial expressions, speech or, as in the context of this dissertation, written compositions, this field aims to extract the "opinions, sentiments and emotions" of the subject [350].

The desire to measure public opinion is not new. The practice of voting to assess public opinion on policy dates back to the fifth century Before Common Era, whilst questionnaires emerged as a measurement technique early in the twentieth century [186]. According to a recent review by Mäntylä *et al.* [186], the first academic paper on measuring public opinion was published in 1940 and was concerned mainly with survey-based techniques. It was, however, not until about 65 years later that research activity in the field started to increase significantly. Figure 4.4 is a timeline of this progression.

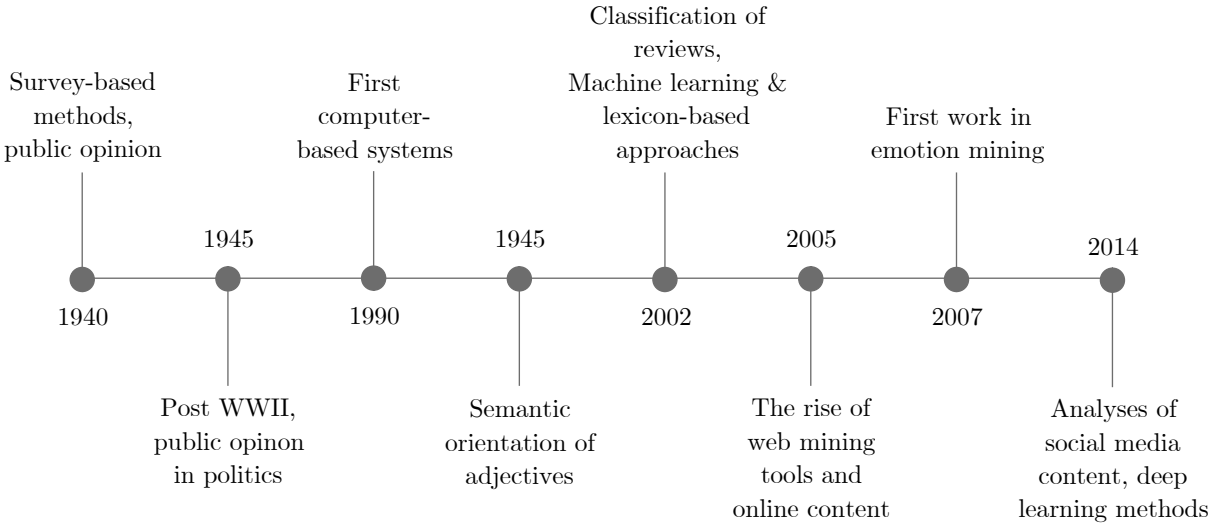


FIGURE 4.4: *Timeline of sentiment analysis research progress.*

After the Second World War, some papers were published addressing the public opinion of those civilisations which had suffered during the war [1, 85, 162]. With the Internet Revolution

of the 1990s came a new body of research based on the use of computer-based systems [122]. Wiebe [339], for instance, published work on the computer-aided detection of subjective sentences. The effect of this new technology on the growth of the field, however, was still insignificant [186].

As per the findings of Mäntylä *et al.* [186], the foundations of modern sentiment analysis were laid during the following years leading up to 2004. The first influential paper was published in 1997 by Hatzivassiloglou and McKeown [117], who developed a method for predicting whether two adjectives selected from a corpus were of the same or of a differing semantic orientation based on the conjunctions which connect them.

Five years later, two well-known papers were published under similar titles, namely *Thumbs up?: Sentiment classification using machine learning techniques* by Pang *et al.* [232], and *Thumbs up or thumbs down?: Semantic orientation applied to unsupervised classification of reviews* by Turney [321]. These papers remain two of the top-cited sentiment analysis papers on Google Scholar to date [186]. As can be deduced from the title, Pang *et al.* used different feature representations and supervised machine learning algorithms to classify movie reviews as either *positive* or *negative*. Turney, on the other hand, determined the polarity of reviews from different domains by averaging the values of the semantic orientations of specific phrases extracted from each review. These semantic orientations were determined by an unsupervised, corpus-based method. The details of these algorithms are described in a following section.

By 2005, 101 papers had been published on sentiment analysis — a steady increase from the 37 papers which appeared until the year 2000. Five years later, this number increased more than tenfold to 1 039. This rapid growth was influenced to a large degree by the increase in opinionated texts on the Internet, as well as the increased availability of *web mining* tools to extract this information [186].

Up to this point, most researchers focused on determining whether words or paragraphs were *positive* or *negative*. In 2007, the first paper was published that considered expanding this classification to several categories of *emotions*. In this paper, news pieces were analysed in order to predict which emotional states were evoked in the readership [253].

The recent upsurge of social media and the associated multitude of highly opinionated, influential and publicly available posts contributed to an even more rapid growth of the field. The number of published papers reached almost 7 000 in 2016, with 99% of these appearing after 2004 [186]. After the success of deep learning methods in other fields, more researchers also started to approach the problem of sentiment analysis from this angle [358].

4.2.1 Tasks

There is little consensus in the literature with respect to terminology pertaining to the field of sentiment analysis [231]. The term *subjectivity* is widely accepted to mean the degree to which something is influenced by personal feelings, tastes or opinions, as opposed to *objectivity*, which is based on factual data and is unaffected by bias. The Merriam-Webster dictionary defines an *opinion* as "a view, judgement, or appraisal formed in the mind about a particular matter," whilst a *sentiment* is "an attitude, thought, or judgement prompted by feeling" [191, 192]. Although the terms *opinion mining*, *polarity classification* and *sentiment analysis* differ slightly in their meaning and focus, they are often used synonymously in the literature [111, 176, 255]. In this dissertation, however, a clear distinction is made between these concepts.

A taxonomy of sentiment analysis by Yadollahi *et al.* [350] is shown in Figure 4.5. As is evident from the figure, opinion mining is a subtask of sentiment analysis and opinion polarity

classification is, in turn, a subtask of opinion mining. Sentiment analysis differs from opinion mining in that it is not only concerned with the positivity or negativity of an observation, but also includes the study of *emotions*, such as *joy* or *anger*. This gives rise to a second subtask of sentiment analysis, namely *emotion mining*.

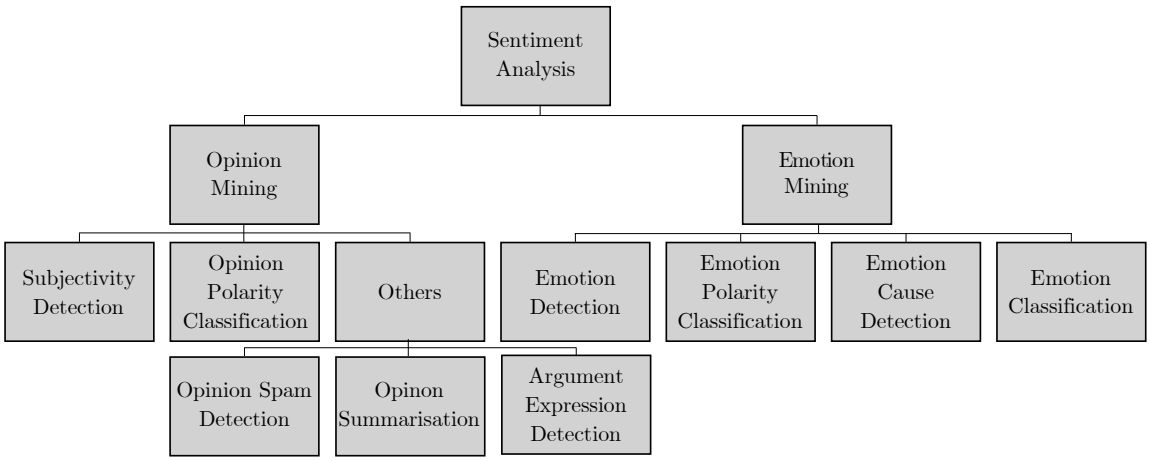


FIGURE 4.5: *A taxonomy of sentiment analysis tasks [350].*

Two prevalent tasks commonly pursued in the realm of opinion mining are *subjectivity detection* and *opinion polarity classification*. The former is concerned with detecting whether a text is *objective* or *subjective*. The text segments "the sky is blue" and "I like the colour blue" are examples of these respective categories [175]. Opinion polarity classification, on the other hand, aims to classify a text as expressing a *positive* or *negative* opinion. In some cases, a third class, *neutral*, is added [350].

Especially in the context of online reviews, it is often found that some users write fake opinions intended to influence the popularity of the target in question. *Opinion spam detection* is the task of identifying such falsifications [350].

The task of *opinion summarisation*, as the name suggests, entails summarising many, possibly differing, opinions towards the same topic. This is particularly useful when decisions are to be made based on the opinions of others [350].

Argument expression detection is concerned with identifying argumentative structures within a text [350]. In other literature, these argumentative structures are referred to as *discourse structures* or *discourse information* [43, 190, 242]. An *adversative structure*, for example, contains words such as *"but," "however"* or *"even though,"* which connect phrases of opposite polarities [242].

Although a clear distinction can be made between the above-mentioned opinion mining tasks, they can complement each other and are therefore often used in combination. It is common, for example, to first classify a text segment as objective or subjective in order to classify the polarity of an opinion based only on its subjective content [322]. Similarly, the identification of various discourse structures can be used to improve performance in the opinion classification tasks. It has, for example, been found that the second argument in an adversative structure usually dominates the first in determining the polarity of the entire phrase. Using such information may make it easier to classify the first of the following sentences as a positive opinion, and the second as a negative opinion [242]:

1. *The car is expensive, but nice.*
2. *The car is nice, but expensive.*

The tasks in the field of emotion mining are defined in a similar fashion. *Emotion detection* entails identifying whether or not an emotion is conveyed in a text, whilst *emotion polarity classification* categorises existing emotions as either *positive* or *negative* in polarity. A more fine-grained analysis is conducted in *emotion classification*, which aims to assign existing emotion in a text to one or more of a predefined set of emotions, such as *joy*, *sadness* or *anger*. Lastly, the task of *emotion cause detection* is concerned with identifying factors which evoke certain emotions [350].

4.2.2 Levels of granularity

In this dissertation, the focus will be on the task of opinion polarity classification. The description of this task may be further refined using the following definition of an *opinion* by Liu and Zhang [178]:

> An opinion [...] is a quintuple, $(e_i, a_{ij}, o_{ijk\ell}, h_k, t_\ell)$, where e_i is the name of an *entity*, a_{ij} is an *aspect* of e_i, $o_{ijk\ell}$ is the *orientation* of the opinion about aspect a_{ij} of entity e_i, h_k is the *opinion holder*, and t_ℓ is the time when the opinion is expressed by h_k.

If an opinion is expressed on the entity as a whole, the aspect GENERAL is used in place of a, which is henceforth denoted by G. Opinion polarity classification, then, entails classifying the opinion polarity $o_{ijk\ell}$ of a given opinion as either *positive*, *negative* or, if applicable, *neutral* [178].

This analysis can be done on several levels, namely the *document level*, the *sentence level* or the *aspect level* [178, 190, 350]. At the aspect level, the analysis objective is to find every quintuple $(e_i, a_{ij}, o_{ijk\ell}, h_k, t_\ell)$ in a given document $\boldsymbol{d}$. Opinion polarity classification at the document level, on the other hand, means determining o on aspect G in the quintuple (e, G, o, h, t), given an opinionated document $\boldsymbol{d}$. It is assumed, in this case, that the document is evaluating a single entity e, and that h and t are unknown or irrelevant [178]. This is the level of analysis adopted in this dissertation. The analysis at sentence level can be viewed as a special case of the document-level analysis, where the document consists of a single sentence or where the opinion polarity of each sentence of a document is determined separately.

Opinion in the definition above refers to a so-called *regular opinion*. There is a second type of opinion, namely the *comparative opinion*. This type of opinion compares multiple entities (*e.g. "Audi makes better cars than Toyota"*), whilst regular opinions are concerned with only one entity or aspect thereof (*e.g. "Audi makes good cars"*) [178, 176].

4.2.3 Analysis approaches

Solution approaches to the opinion polarity classification problem may be grouped into two major categories, namely *lexicon-based* approaches and *machine learning* approaches (or *statistical* approaches) [111, 190, 255]. The former are often also referred to as *knowledge-based* approaches, since they rely on existing semantic resources such as *sentiment lexicons*[5] [93, 111]. The latter approaches, on the other hand, are concerned with learning *patterns* based on labelled, historical data, without the use of any additional resources [93].

[5] A *sentiment lexicon* provides information on the *prior polarity* of a word, *i.e.* whether it is considered *positive*, *negative* or *neutral* in most contexts [160].

There is little in the literature on the classification of sentiment analysis techniques beyond these two categories. One significant[6] paper on this matter was published by Medhat *et al.* [190] in 2014, in which the taxonomy shown in Figure 4.6 was proposed. This categorisation is used as a road map for the discussion of the literature in this section, and is critically evaluated in the process.

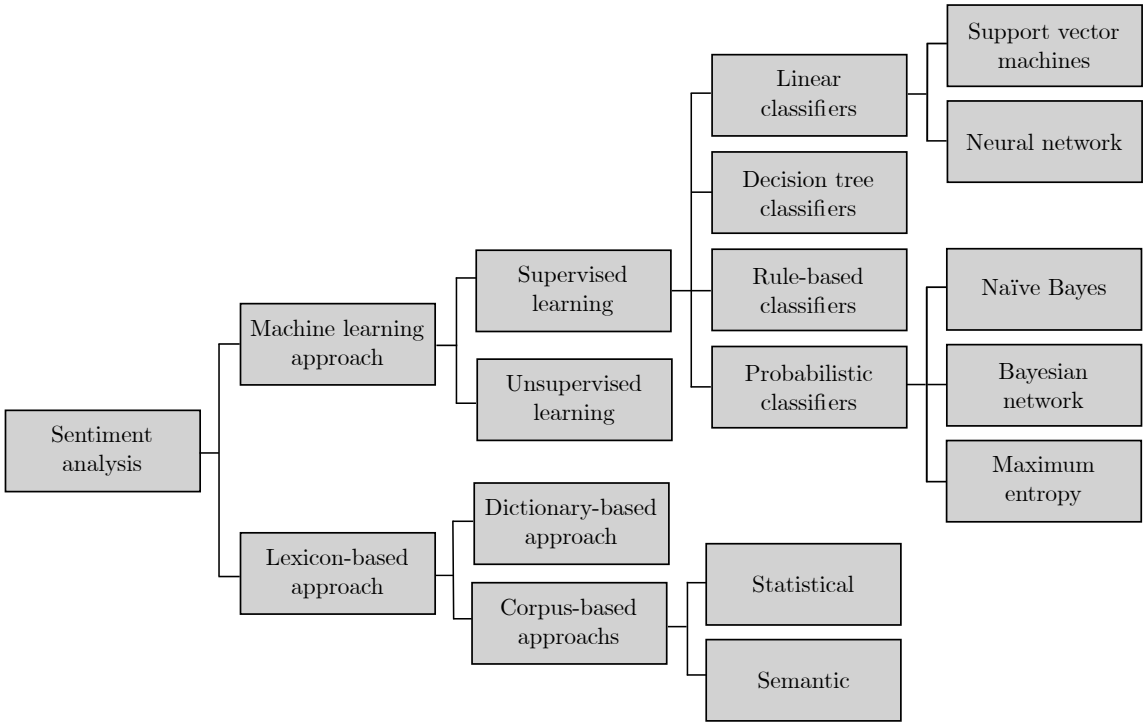


FIGURE 4.6: *A taxonomy of sentiment analysis techniques proposed by Medhat et al. [190].*

Lexicon-based approaches

The family of lexicon-based approaches is aimed at determining the sentiment polarity[7] of a piece of text by counting and weighting the *known* polarity of individual words and phrases contained in the text [28, 70, 111]. In the most simple approach, the number of negative words contained in a text is subtracted from the number of positive words. The text is then classified as *positive* if its total score is greater than zero, or *negative* otherwise [70]. Other scoring functions can also be used. One may, for example, calculate the score for an entity e described in a given text as

$$\text{score}(e) = \sum_{w_i \in \mathcal{S} \cap \mathcal{V}} \frac{S(w_i)}{d(w_i, e)},$$

where w_i is the opinion word, $\mathcal{V}$ is the set of all opinion words included in the lexicon, $\mathcal{S}$ is the set of all words in the text and $S(w_i)$ and $d(w_i, e)$ are the semantic orientation of word i and the distance, measured in number of words, of word i from entity e, respectively. In

[6]The paper had 731 Google Scholar citations at the time of writing this dissertation, amounting to approximately 183 citations per year. In comparison, this value ranged from 179–613 citations per year for the ten top-cited papers on sentiment analysis on Google Scholar in 2016 [186].

[7]The terms *sentiment polarity*, *opinion polarity* and *semantic orientation* are used interchangeably this dissertation to describe whether a word, phrase or text conveys *positive*, *negative* or *neutral* sentiment.

this way, opinionated words which are further away from the entity are given a lower weight in determining the final score [28, 75].

Lexicon-based methods may further be differentiated by the way in which the sentiment lexicons are created. There exist three general approaches for lexicon generation, namely the manual approach, the *dictionary-based* approach and the *corpus-based* approach. The former is time-intensive and is therefore typically used in conjunction with the latter two (automatic) methods. More specifically, a small set of *seed words* is manually labelled with their semantic orientation, and this set is then expanded using automatic methods [111, 190].

Dictionary-based approaches expand the set of seed words by using existing thesauri or other available resources, such as *WordNet*,[8] to find their synonyms and antonyms. Given the fact that words generally have semantic orientations that are of the same orientation as their synonyms, and of opposite orientation to that of their antonyms, new words may be added to the lexicon iteratively, until no new words are found [129]. The advantage of this approach is its simplicity. It has a significant disadvantage, however, in that it cannot recognise the domain-specific polarity or the *contextual polarity* of words [111, 190]. Some words, such as *"warm,"* may generally be considered positive, and therefore have a positive *prior polarity*. The same word may, however, be used to convey a negative sentiment in other contexts (*e.g. "warm beer"*) [319]. Similarly, a word may carry a positive connotation in one domain, but a negative one in another, as mentioned in the discussion of topic-related features in §4.1.2. A *"long battery life"* for an electronic device, for example, is certainly favourable, whilst a *"long queue"* in the service industry is not.

The problem of contextual polarity is associated with both dictionary-based and corpus-based approaches. Lexicon-based methods are therefore often combined with other methods in order to address these shortcomings. Qiu *et al.* [250], for instance, used syntactic parsing and a set of manually constructed rules to classify contextual polarities correctly. A rule-based approach was also followed by Ding *et al.* [75], who used POS-tagging along with *negation*, *conjunction* and *synonym-antonym rules* to create a *"holistic"* lexicon-based approach. Rule-based techniques appear in the taxonomy in Figure 4.5 only in association with the machine learning approach. The discussion of their placement in the categorisation of sentiment analysis techniques is revisited in that respective section.

On the other hand, the problem of domain-specific polarity can be alleviated by means of corpus-based lexical methods if the documents in the corpus are selected exclusively from the domain in question [111]. In this approach, the lexicon is expanded based on syntactic or co-occurrence patterns in a large corpus [111, 190].

One of the first of these methods was published by Hatzivassiloglou and McKeown [117]. Their approach leveraged the fact that linguistic constructs impose constraints on the polarities of their arguments. Conjunctions, such as *"and"* and *"but,"* typically constrain the adjectives joined by them to be of the same and of a different semantic orientation, respectively. Similarly, morphological relationships between adjectives can also impose constraints. Semantic orientations of adjectives transformed in a particular manner, such as *adequate–inadequate* or *thoughtful–thoughtless*, typically differ [117]. Using these constraints, a *log-linear regression model* was constructed to predict the dissimilarity $y \in (0, 1)$ of the semantic orientation of a pair of adjectives based on their usage in a corpus. More specifically,

$$y = \frac{e^{\eta}}{1 + e^{\eta}},$$

where $\eta = \boldsymbol{w}^T \boldsymbol{x}$, and where $\boldsymbol{x}$ is a vector containing the frequency with which each of a given set

[8] *WordNet* is a lexical database for English in which words are organised into synonym sets ('*synsets*'), each representing one underlying concept, and these sets are interlinked by semantic and lexical relations [196].

of conjunctions was observed between the pair of adjectives in the corpus and $\boldsymbol{w}$ is a vector of weights to be learnt during training. A graph was then constructed using the predicted level of dissimilarity between each pair of variables (setting this value to 0.5 for adjective pairs with no observed conjunctions). Subsequently, a clustering algorithm was used to separate the adjectives into two distinct groups. Finally, the group with higher average frequency of use in the corpus was assigned the label *positive*, based on the results of previous research by Hatzivassiloglou and McKeown [117] who found this to be the general case.

Another well-known corpus-based method is the *Semantic Orientation from Association* (SO-A) strategy published by Turney [321]. The idea behind this method is that the semantic orientation of a word or phrase may be inferred from its statistical association with sets of positive and negative seed words. Mathematically, this may be expressed as

$$S(x) = \sum_{p \in \mathcal{P}} A(x, p) - \sum_{n \in \mathcal{N}} A(x, n),$$

where $\mathcal{P}$ and $\mathcal{N}$ are the sets of positive and negative seed words, respectively, $A(x, y)$ is a measure of association between a word or phrase x and a seed word y, and $S(x)$ is the semantic orientation of x. This association measure can be determined using either LSA or *point-wise mutual information* (PMI). When the former is used, association between words is calculated as the dot product of their vector representations in LSA space, as described in §2.3.2. When the latter is used, however, the PMI is estimated using the number of *hits* (matching documents) returned when issuing a search query to a search engine. In the *Alta Vista Advanced Search Engine* employed by Turney, the *near* operator returns documents in which arguments occur within 10 words of one another. The semantic orientation of a word or phrase x may then be calculated as

$$S(x) = \log_2 \left(\frac{\prod_{p \in \mathcal{P}} hits(x \text{ near } p) \prod_{n \in \mathcal{N}} hits(n)}{\prod_{n \in \mathcal{N}} hits(x \text{ near } n) \prod_{p \in \mathcal{P}} hits(p)} \right). \tag{4.1}$$

The expression in (4.1) is a *log odds ratio* over all positive and negative seed words. This corresponds to the logarithm of the ratio of the proportion of documents containing the positive word that also contained the given phrase and the proportion of documents containing the negative word that also contained the given phrase [322]. In the original paper, the PMI was used along with only one positive and one negative seed word (*"excellent"* and *"poor"*, respectively) to calculate the semantic orientations of phrases that exhibited certain parts-of-speech patterns. In experiments with 410 different reviews, the algorithm attained an average accuracy of 74% with the accuracy for bank and automobile reviews ranging from 80–84% [321].

Medhat *et al.* [190] further classified corpus-based approaches as either *semantic* or *statistical*. This distinction was proposed in an earlier survey by Tsytsarau and Palpanas [319]. The SO-A method described above may be viewed as a typical example of a statistical corpus-based approach, where words are assigned a polarity based on their statistical similarity (co-occurrence) with positive and negative seed words. According to Tsytsarau and Palpanas [319], the semantic approach is distinguished from this in that the similarity of words is not computed using statistics, but is rather determined based on certain principles which govern how "semantically close" a set of words is.

One may argue, then, that the algorithm developed by Hatzivassiloglou and McKeown [117] is an example of such a method, since polarities are assigned according to linguistic constructs which encode semantic information. Both Medhat *et al.* [190] and Tsytsarau and Palpanas [319], however, refer to the use of *WordNet* to determine the sentiment polarity of a new word as an example. A set of seed words may, for instance, be expanded using synonym and antonym

relationships, and the semantic orientation of an unknown word in the corpus may then be determined as the ratio of the number of its positive synonyms to the number of its negative synonyms [190]. This approach, however, more closely resembles the dictionary-based approach described earlier. Other examples mentioned by the authors make use of more complicated relationships, including path lengths between words in a graph encoding the synonymy relations of words [142]. Whilst these approaches certainly make use of *semantic* relationships, they do not make use of the *corpus* at all when creating the lexicon. It is therefore not clear why these methods should be classified as *corpus-based*.

Aside from these two papers (and direct citations) no other references could be found for partitioning corpus-based approaches into statistical and semantic categories. Ravi and Ravi [255] proposed instead to partition approaches to lexicon creation in general as either *ontology*[9]*-based* or *non-ontology-based*. This distinction, however, refers more to the structure of the final sentiment lexicon itself rather than to the way in which it is constructed, differing considerably from the classification by Tsytsarau and Palpanas [319]. Due to the lack of consistency in the literature, a categorisation of lexicon-based methods beyond dictionary-based and corpus-based approaches is not advocated in this dissertation.

Machine learning approaches

Opinion polarity classification may be framed as a regular (text) classification problem. More specifically, each document represents a record, defined by a set of features and a target class (in this case, its sentiment polarity). A set of labelled documents can then be used to train a model to predict the class label for a document instance of an unknown class. Consequently, any existing machine learning algorithm for classification can be applied to this problem [178, 176, 190]. The focus in this section falls on the algorithms used to predict a target class based on the features of the input data. It is assumed that the unstructured data have been transformed to a vectorised format according to the procedures described in §4.1.

In the literature, machine learning is predominantly applied to the sentiment analysis problem in a supervised setting, in which all training samples are annotated [201]. Approaches also exist for the case where only some samples have labels, as well as for a completely unsupervised setting.

Medhat *et al.* [190] further categorised supervised learning algorithms as *linear classifiers*, *decision tree classifiers*, *rule-based classifiers* or *probabilistic classifiers*, as shown in Figure 4.6. This creates the impression that these algorithms are of equal importance in the literature, in contrast to the findings of other literature surveys, which suggest that most researchers "agree on the learning techniques" of naïve Bayes, SVM and often also maximum entropy [201, 316, 331]. These algorithms, described in §3.3.4, §3.3.3 and §3.3.6, respectively, were the three algorithms initially employed by Pang *et al.* [232] and may therefore be referred to as *traditional* methods in the context of sentiment analysis. In recent surveys, deep ANNs have also been recognised as an important class of algorithms [201, 350, 358]. *Neural networks* are included in the taxonomy in Figure 4.5 as linear classifiers alongside SVMs. Whilst this designation may have been appropriate for earlier work, most network architectures now comprise several hidden layers and non-linear activation functions, and can therefore no longer be described as linear classifiers.

Although other algorithms, such as decision trees (described in §3.3.2) and Bayesian networks in Figure 4.5, have indeed been used by some researchers in the realm of sentiment analysis,

[9] An *ontology* is a formal representation of knowledge, capturing semantic associations between concepts and relationships [255].

these methods are not dominant in the literature. Medhat *et al.* [190] themselves noted, for example, that Bayesian networks are not widely used in text mining due to their computational complexity [190]. In order to better reflect the current state of the literature, it may be more appropriate to classify such algorithms as a separate group of additional methods.

It is, therefore, proposed that the current subcategories of supervised learning in Figure 4.5 should be replaced by three new categories, namely *traditional*, *deep neural networks* and *other* algorithms. Furthermore, the *unsupervised learning* category should be extended to include semi-supervised and weakly supervised methods, as reflected in the detailed description of the taxonomy by Medhat *et al.* [190].

Moreover, as mentioned in the previous section, rule-based classifiers are also often used in combination with lexicon-based methods. Although some of these classifiers make use of learning algorithms to discover an appropriate set of rules, such as the one employed by Walker *et al.* [329], these rules are often also manually defined. Qiu *et al.* [250], for instance, made use of pre-specified rules to extract topic words of opinionated sentences. In light of this, it is deemed more appropriate to view rule-based methods as a separate approach from lexicon-based and machine learning approaches, and to rather indicate that these three methods are often used in combination. Devika *et al.* [70] also classified approaches to the sentiment analysis problem into these three proposed categories (lexicon-based, rule-based and machine learning approaches).

The revised taxonomy of sentiment analysis techniques is shown in Figure 4.7. In the remainder of this section, each of the new subcategories of the machine learning approach is discussed in more detail.

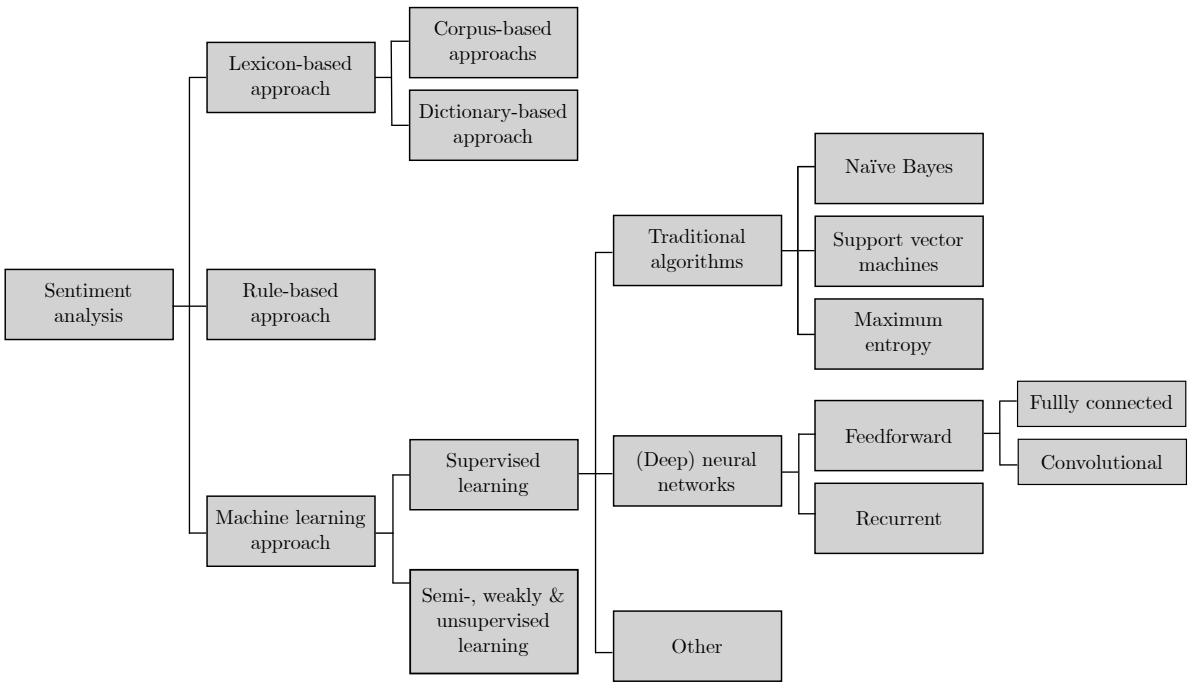


FIGURE 4.7: *A revised taxonomy of sentiment analysis techniques.*

In general, machine learning techniques have been shown to outperform lexicon-based methods [6, 49, 111], in some cases with the best performing lexicon-based classifier achieving an accuracy that is approximately 10% lower than that of the worst performing machine learning algorithm and 20% lower than the best performing machine learning algorithm [6]. Dhaoui *et*

al. [72] found the performance of both approaches to be similar in terms of the F-measure, although the accuracy achieved by the machine learning approach still proved to be superior.

In the first implementation of *traditional* machine learning algorithms for sentiment analysis, Pang *et al.* [232] found that SVMs yielded the best results in terms of accuracy, followed by naïve Bayes and maximum entropy. Furthermore, the best accuracy was achieved using only the presence of unigrams as features. These methods, and particularly the first two, are frequently used as baseline methods for sentiment analysis [331]. Often, they still outperform more recent and more sophisticated models [280, 331], with improvements from the baseline originating largely from efforts in feature engineering and feature selection [93, 316].

Sharma and Dey [280] compared the performance of the traditional algorithms with four *other* supervised learning algorithms, including the kNN algorithm (described in §3.3.1) and a C4.5 decision tree. They found the traditional algorithms to be superior by a significant margin in terms of accuracy. Wang *et al.* [330] used the same algorithms (naïve Bayes, SVM, maximum entropy, kNN and decision trees) as base learners in three ensemble methods and found a performance increase over the base learners by using ensembles. Other approaches include hybrid methods combining SVM with the *particle swarm optimisation* heuristic or combining a *Markov blanket-directed acyclic graph*, which captures conditional dependencies between words, and the *tabu search* metaheuristic for improved accuracy [14, 20].

In 2009, Bengio [23] noted, based on research results at the time, that in order to learn complicated functions capable of representing high-level abstractions, such as vision and language, deep learning architectures are necessary. As indicated in Figure 4.7, feedforward and recurrent network architectures, both of which are described in detail in §3.5.3, may be used for sentiment classification. Approaches also exist in which *recursive* networks are employed. This is, however, only common for sentence-level and aspect-level sentiment classification [358], and therefore falls outside of the scope of this dissertation. Salient examples of each of the former two types of network architecture are given next.

Moraes *et al.* [201] compared the performance of popular methods of naïve Bayes and SVM with that of an ANN. They used a bag-of-words representation for all three algorithms and a feedforward neural network with a single, fully connected, hidden layer. The results indicated that the ANN-based classifier outperformed SVM by a statistically significant margin in most experiments. The naïve Bayes classifier did not achieve comparable performance. The study also revealed a disadvantage of the ANN in the large increase in its training time as the size of the vocabulary increases.

Bespalov *et al.* [27] developed a CNN for sentiment classification. This architecture is shown in Figure 4.8. In the first stage, the network creates word embeddings from a *one-hot-encoding* representation (where each word is identified by an index in the vocabulary). Subsequently, n-gram phrase embeddings are generated using concatenated word embeddings as input. In the example in the figure, trigrams are used. This is the convolutional layer of the network, in which the filter traverses the sentence in the order in which it was written, considering n words at a time. The phrase embeddings are then averaged to yield a document embedding. This is equivalent to *average pooling*. Finally, the document vector is fed into a typical classification layer, which returns a prediction for the sentiment polarity.

Later papers in which a similar hierarchical method was adopted to construct document representations refer to the *principle of semantic compositionality* or *Frege's principle* as the foundation for this approach, which states that the meaning of a complex whole (*e.g.* a document) is a function of the meaning of its parts (*e.g.* the words it contains) and the way in which these parts were combined (*e.g.* the word order) [237, 308].

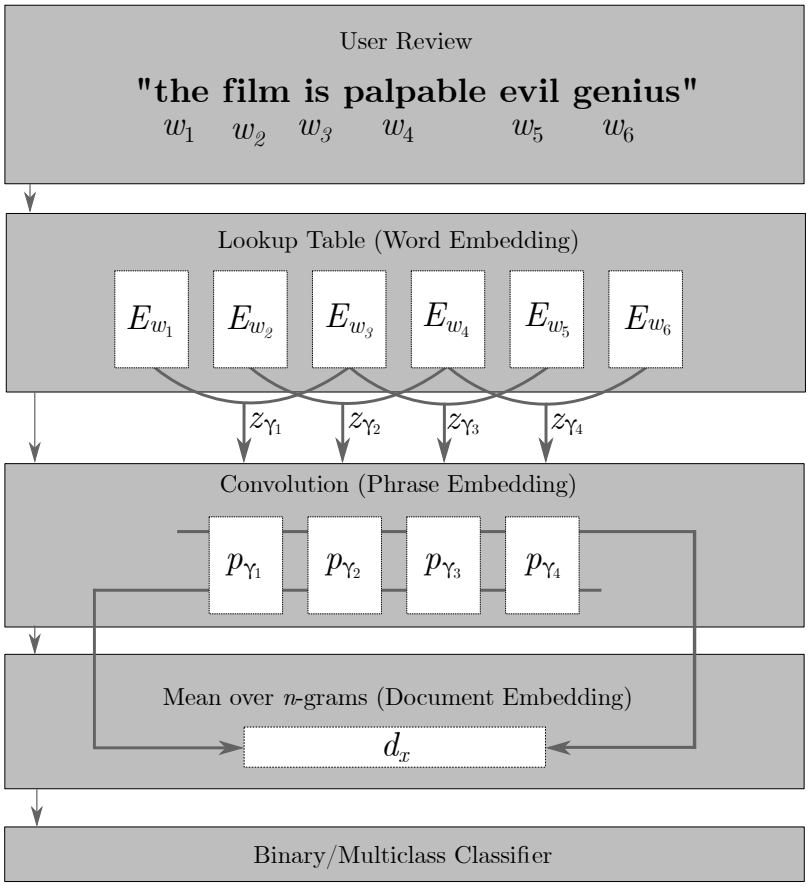


FIGURE 4.8: *CNN architecture for sentiment classification (Adapted from Bespalov et al. [27]). Word embeddings are generated from a one-hot-encoded word representation to be taken as input for a convolutional layer. The output of the convolutional layer is then averaged to yield a document embedding, which serves as input for a classification (output) layer returning a sentiment prediction.*

Using a convolutional architecture instead of the conventional bag-of-words model preserves the original word order. Furthermore, training embedding feature learning and sentiment classification *end-to-end*[10], ensures that the learnt embeddings are useful for the classification task.

Bespalov *et al.* [27] found that their model was able to outperform SVM-based models when provided with a large enough training data set. Furthermore, the model trained by means of unigrams matched the performance of a bag-of-words perceptron model trained on unigrams and bigrams (thereby using a larger dictionary) for binary classification, and was able to outperform these baseline methods in a multi-class context.

Johnson and Zhang [140] adopted a similar approach in what they referred to as *seq-CNN*. One notable difference between their approach and that of Bespalov *et al.* [27] is a removal of the word embedding layer. Instead, the phrase embeddings are learnt directly from the concatenation of the one-hot-encoded word vectors. This reportedly leads to performance increases whilst also reducing the number of trainable parameters. In the same paper, Johnson and Zhang [140] also proposed two alternative CNN architectures, namely *bow-CNN* and a parallel configuration combining up to three convolutional layers. The *bow-CNN* configuration reduces computational

[10] *End-to-end* here means that both tasks are trained simultaneously, as opposed to first learning feature representations and then training the classifier based on these representations [98].

cost by reducing the dimensionality of the phrase vectors. If, for example, the vocabulary in a corpus is $\mathcal{V} = \{\text{“}don't\text{”}, \text{“}hate\text{”}, \text{“}I\text{”}, \text{“}it\text{”}, \text{“}love\text{”}\}$, the one-hot encoded format for the phrase “I love it,” given by

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

would be replaced by $\begin{bmatrix} 0 & 0 & 1 & 1 & 1 \end{bmatrix}$ in the alternative bag-of-words representation used in bow-CNN. Since the bag-of-words model is applied to each successive n-gram and not to the entire document, this does retain some information pertaining to the word order. In the parallel configuration, several convolutional layers with different n-values are applied and their results combined in the output layer, so that different, complementary features can be learnt from each parallel stream in order to improve model accuracy. For all three configurations, the convolutional layer is followed by a pooling layer and a fully connected prediction layer, just as in Figure 4.8. Furthermore, ℓ_2 regularisation, dropout and *response normalisation* (a predecessor of batch normalisation) are applied. These techniques were described in §3.5.1.

The results showed that seq-CNN outperformed bow-CNN for sentiment classification tasks, whilst the converse was true for topic categorisation. Furthermore, these two models achieved competitive, if not superior, performance compared to state-of-the art supervised algorithms. Implementations of the parallel configuration outperformed all other models tested, with the best performance achieved by a network containing three parallel layers — two seq-CNN layers and one layer that regards the entire document as one region (n is chosen as the length of the document) and represents this using a *bag-of-n-grams* model. This suggests that both short word sequences and a global context are useful for sentiment classification [140].

RNNs are particularly popular in natural language processing tasks due to their ability to encode sequential information. By virtue of their ability to better model long-term dependencies, LSTM networks are typically used rather than standard RNNs [358].

Tang *et al.* [308] proposed a slightly different architecture for learning document representations in a sentiment classification context that attempts to encode intrinsic relationships between sentences in a document. First, sentence representations are learnt using either a CNN or an LSTM network and *pre-trained*[11] word embeddings as input. Subsequently, sentences and their relationships are encoded into a document representation using a *gated recurrent neural network*, which is analogous to an LSTM model [308].

Xu *et al.* [347] proposed a variant of the LSTM model that addresses shortcomings of the model in the face of longer documents. By allowing different *memory groups* to have different “*forgetting rates*” (σ-values), the network is able to store both global (low forgetting rates) and local (high forgetting rates) semantic information [358]. Their *Cashed Long Short Term Memory* network outperformed *“vanilla”* LSTM for data sets with documents of longer average length, whilst the performance of both networks was similar for shorter texts [347].

Yang *et al.* [351] proposed the *Hierarchical Attention Network* for document-level sentiment classification, in which an RNN-based model makes use of the attention mechanism (described

[11] *Pre-trained* ANNs are networks that have been trained using one data set and are then employed for a different task. Typically, some parameter values are fixed, while others are *fine-tuned* (trained) on the new data set. This process is called *transfer learning* [224]. In this case, a network is pre-trained to produce word embeddings, which are then taken as input for the LSTM/CNN network.

in §3.5.3) to selectively afford more weight to words and sentences that are thought to contribute more significantly to the overall sentiment of the document.

More complex network architectures incorporate user, product and aspect information or address issues related to cross-lingual or cross-domain sentiment classification [358].

A common problem faced in the machine learning paradigm is a lack of labelled data. Unlabelled data, on the other hand, is typically available on a relatively large scale. In order to take advantage of unlabelled data, unsupervised, semi-supervised or weakly supervised models are implemented. These methods represent the last category of sentiment analysis techniques in Figure 4.7.

Glorot *et al.* [100] employed a network consisting of several stacked denoising autoencoders, as described in §3.5.3, to learn representations for product reviews in an unsupervised manner using all available data (labelled and unlabelled). Subsequently, these representations were taken as input into a supervised SVM model to train a sentiment classifier using only the labelled data. This is similar to the "unsupervised training–supervised fine-tuning" scheme in which autoeconders are used to pre-train network parameters in an unsupervised manner. This approach is effective when the data distribution is correlated with the classification task [23]. In sentiment analysis problems, however, word co-occurence is often not well correlated with sentiment prediction [183]. Many researchers therefore resort to methods in which the information from the available data labels is leveraged during the feature generation step.

Guan *et al.* [109] attempted to do this by using the ratings of online customer reviews as weakly supervised labels for the generation of sentence embeddings, where some sentences typically do not conform to the overall rating. A classification layer was then added for final sentiment prediction in order to fine-tune the network using a smaller sample of correctly labelled sentences.

Zhai and Zhang [357], on the other hand, added a supervised element to the training of the autoencoder. First, labelled data were used to train a linear classifier (SVM), which generates a weight matrix $\boldsymbol{\theta}$. This matrix was then used to adjust the loss function of the autoencoder to

$$L(\tilde{\boldsymbol{x}}, \boldsymbol{x}) = (\boldsymbol{\theta}^T(\tilde{\boldsymbol{x}} - \boldsymbol{x}))^2,$$

effectively adding a heavier weight to features which contribute to the sentiment classification, and ignoring task irrelevant features. With additional scaling of the loss function to compensate for bias in the linear classifier, the model was found to outperform the traditional DAE, as well as baseline models. Furthermore, using additional unlabelled data points in training the autoencoder resulted in performance improvements, confirming that the model is able to successfully leverage additional information from unlabelled data sources.

Alternative approaches to semi-supervised learning include graph-based methods and *wrapper methods* [60]. The former encode similarities between samples into a graph. Labels of annotated samples are then propagated according to these similarities. As previously mentioned, however, sample similarities of text documents typically reflect topic distributions rather than the sentiment information that is desired. Wrapper methods use supervised learning algorithms functioning in an iterative fashion and include *self-training* and *co-training*. In principle, self-training methods are trained on labelled data and predict labels for unlabelled data. Where a high confidence level is achieved for a prediction, the associated unlabelled sample is added to the labelled training set with its predicted class label. This procedure is iterated until no further unlabelled samples exist. Co-training works in a similar fashion, except for the fact that two supervised models are trained in each iteration using two distinct feature sets and the most confident predictions are selected from both classifiers.

In an unsupervised approach to sentiment classification, Li *et al.* [172] added a *sentiment layer* to the LDA topic model described in §2.3.3. By modelling topic and sentiment concurrently, the dependency of the sentiment polarity of certain words on their context or domain is captured. *Sentiment-LDA* assumes that, in addition to the distribution over topics, a distribution over sentiment labels for each topic is randomly selected for each document. Each selected word in a document then depends on the randomly selected topic, as well as the sentiment label drawn from these distributions. The authors also proposed a further extension to the model called *Dependency-Sentiment-LDA*, in which dependencies between sentiment labels in a document are modelled by means of *Markov chains*. Each sentiment label is assumed to depend on the label of the previous word, in line with the argument that humans express sentiment in a coherent manner. The model parameters were initialised using sentiment lexicons during experiments and the approach was found to outperform purely lexicon-based methods making use of the same lexicons. Furthermore, the Sentiment-Dependency-LDA model achieved an only slightly lower accuracy than known supervised methods in respect of the same data set [172].

The performance of machine learning methods depends highly on the model variant, features and data set used [280, 331]. It is therefore advisable to test several configurations on a given data set in order to find one that is most suited for that particular scenario.

Ensemble learning and hybrid approaches

As mentioned in §3.4, there has been a growing interest in ensemble learning techniques aimed at producing more generalisable models. Compared to other research areas, however, the use of ensembles in sentiment classification is still limited [330]. Nevertheless, several researchers have shown that ensembles and hybrid models perform better than their individual components in respect of sentiment classification.

Wilson *et al.* [341], for instance, observed a 23% to 96% increase in classification accuracy when using boosting compared to individual classifiers. Prabowo and Thelwall [247] similarly found that a hybrid approach combining several lexicon-based and rule-based approaches with an SVM classifier outperformed all individual classifiers in three of four tested cases.

Typical ensemble learning approaches towards sentiment analysis include stacking and weighting of different types of classifiers. Comparatively, meta-learning approaches appear to outperform simple weighting approaches towards sentiment classification. Omar *et al.* [220] compared the performance of a simple voting ensemble with stacking ensembles employing naïve Bayes, logistic regression and SVM as meta-learners, respectively, for classifying sentiment in Arabic customer reviews. They found that all ensemble techniques outperformed their individual component models, and that stacking with logistic regression produced the best results in terms of the F-measure for both subjectivity and sentiment classification. Tsutsumi *et al.* [318] compared the performance of an SVM, ME and a lexicon-based approach with ensembles of these classifiers employing simple voting, weighted voting and meta-learning by means of an SVM, respectively. Once again, all ensemble classifiers were found to outperform the individual base learners, whilst the best performance was achieved by stacking with SVM as the meta-learner.

Wang *et al.* [330], however, noted difficulties associated with selecting base learners for a stacking approach and actively turned away from such methods. Instead, they compared bagging, boosting and *random subspace* or vertical partitioning (see §3.4) approaches towards ensemble learning with naïve Bayes, ME, a decision tree, kNN and an SVM as base learners, respectively. The results showed that the random subspace approach with an SVM as the base learner performed best in six out of ten cases, and comparably with the best results in the other four cases.

Other researchers have achieved good results by combining stacking with input data manipulation in respect of the feature dimension. Xia *et al.* [346], for example, compared the effectiveness of various ensemble configurations. More specifically, they employed two different feature sets (POS-related features and features capturing word relations) and three different base learners (maximum entropy, naïve Bayes and SVM) in conjunction with three different ensemble methods (simple voting, simple averaging of output probabilities and stacking). The stacking approach was implemented in several different configurations using linear regression, logistic regression, a neural network and SVM as meta-learners, respectively, and in the form of the *StackingC* approach with linear regression as a meta-learner. In general, Xia *et al.* [346] found that constructing an ensemble based on the different feature sets outperformed the use of a joint feature set, and that ensembles of different classifiers outperformed individual classifiers, with meta-learning approaches faring better than simple voting or averaging. Furthermore, ensembles of both different feature sets and different classification algorithms outperformed the associated individual approaches. Araque *et al.* [8] similarly found that an ensemble of classifiers trained in respect of different types of features (surface features, generic word embeddings and word embeddings trained for sentiment analysis) performed best and that the meta-learning approach outperformed simple majority voting.

More complex approaches involved the use of additional model selection layers. Hassan *et al.* [116], for example, developed a two-stage ensemble framework for sentiment analysis of Twitter data. During the first stage, different data sets, feature sets and classification algorithms are combined to form an array of different models. A subset of the possible models is then selected by means of an iterative search algorithm seeking to maximise validation performance of the ensemble. This approach was shown to better identify extremities in the sentiment expressed in Tweets over time. On a similar note, Khan *et al.* [154] proposed a semi-supervised framework in which SentiWordNet scores were revised by means of PMI, and where feature weights for these scores were learnt by an SVM-based *multi-objective model selection* algorithm, resulting in superior performance in respect of benchmark data sets over existing lexicon-based and machine learning approaches.

Finally, Lam and Kai [166] proposed a meta-learning ensemble model whose input features represent meta-data on the data set or task at hand rather than the outputs of base learners. The model then recommends a specific algorithm to make the prediction based on the given situation. This approach therefore represents a mixture of expert models (see §3.4). Their results showed an improvement in classification accuracy for the ensemble over individual base learners.

4.2.4 Summarising sentiment

The preceding section was concerned with determining the sentiment polarity of a given text document. The purpose of most opinion mining systems, however, is to represent the overall sentiment, or several common trends, contained in a collection of such documents. It is therefore necessary to create a summarised view of the results obtained by applying the methods described in §4.2.3.

Multi-document summarisation in the realm of sentiment analysis is often conducted by adapting standard methods from topic-based summarisation in order to produce *textual summaries*. These typically fall into one of two categories, namely *template instantiation* and *passage extraction* [129]. The former entails extracting certain entities and features from the corpus and generating natural language to describe the sentiment regarding these according to some predetermined template. The latter is concerned with selecting sentences or segments of text according to certain importance criteria and displaying these to the user.

Carenini *et al.* [45] implemented both approaches in an attempt to summarise a corpus of evaluative documents. They first extracted information from each document, namely the *features* of the entity that were evaluated, the *polarity* of the evaluation and its *strength*. Their sentence extraction method is based on the existing topic-based framework *MEAD* [252] and extracts sentences according to the number of features mentioned, as well as the strength of the polarity associated with each expressed opinion. The natural language generation approach applies a similar importance metric to each feature in the corpus (where a high importance is associated with frequently mentioned or strongly evaluated features) and relays information to the user regarding the average sentiment about this feature. Furthermore, the dispersion of the sentiment is also considered. Both positive and negative feedback is provided to the user when opinions about a feature are located on either side of the spectrum with approximately equal frequency.

Hu and Liu [129] proposed a more structured *aspect-based summary*, an example of which is shown in Figure 4.9. The summary for a particular entity is given by an indication of the number of positive and negative reviews of each of its features. Furthermore, the user is able to follow a link, denoted by "<individual review sentences>," in order to view individual sentences extracted from the corpus which express a positive or negative opinion. In order to generate such a summary, product features are identified by applying a POS-tagger to the reviews in order to identify nouns or noun phrases, subsequently selecting those that are mentioned in several reviews. Each sentence is then assigned a sentiment polarity (this may be done using any method described in §4.2.3) and the number of positive and negative sentences containing each of the selected features is determined.

*Digital_ camera_*1:
Feature: **picture quality**
Positive: 253
<individual review sentences>
Negative: 6
<individual review sentences>
Feature: **size**
Positive: 134
<individual review sentences>
Negative: 10
<individual review sentences>
. . .

FIGURE 4.9: *An aspect-based summary by example of reviews of a digital camera [129].*

The structured summary in Figure 4.9 addresses, to some extent, a weakness of most textual summaries. More specifically, a traditional text summary typically contains phrases such as "most people dislike this product." The percentage or number of reviewers in disfavour may, however, be more meaningful to the user [178]. Furthermore, as pointed out by Pang and Lee [231], *visual* or *graph-based* summaries may be more suitable and useful for representing the content of multiple documents.

Particularly when the opinion polarity of a document can be represented by a number or binary value, graphical summaries can be generated based on *summary statistics* of these values, which may include the average sentiment score or the number of favourable and unfavourable reviews. *Bounded statistics*, which fall within a predetermined range of values, may be depicted in terms of size, extent or colour shading [231].

Gamon *et al.* [94], for example, summarise the results of their opinion mining system *"Pulse"* using colour shading to represent sentiment and size to represent frequency counts. A compressed form of the output given in their paper is shown in Figure 4.10. The main area of the figure contains rectangular boxes, each of which represents a particular *topic* discussed in the review and are inscribed with the most salient keyword of that topic. After several failed attempts at using common clustering algorithms to find these topics, Gamon *et al.* [94] devised their own algorithm, which creates N clusters for the N most frequently occurring tokens in a corpus, taking care not to make use of stop words or words with strong sentiment (*e.g "terrible"*), and considering especially words that are known to be important in the domain. The size of a box represents the number of sentences in a given cluster, and the colour indicates the average sentiment of sentences in that box, where dark green indicates a strongly positive sentiment, dark red a strongly negative sentiment and white a balanced or neutral sentiment.

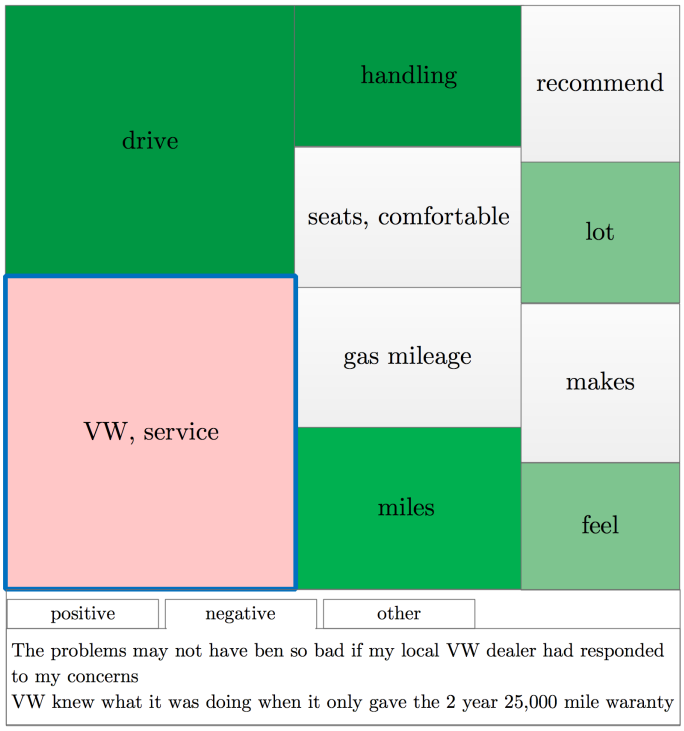


FIGURE 4.10: *An extract of the summary generated by the opinion mining system "Pulse" for automobile reviews. Each box represents a cluster (topic) and its most salient keyword. The size and colour of the box represent the number of sentences and average sentiment in a cluster, respectively (adapted from Gamon et al. [94]).*

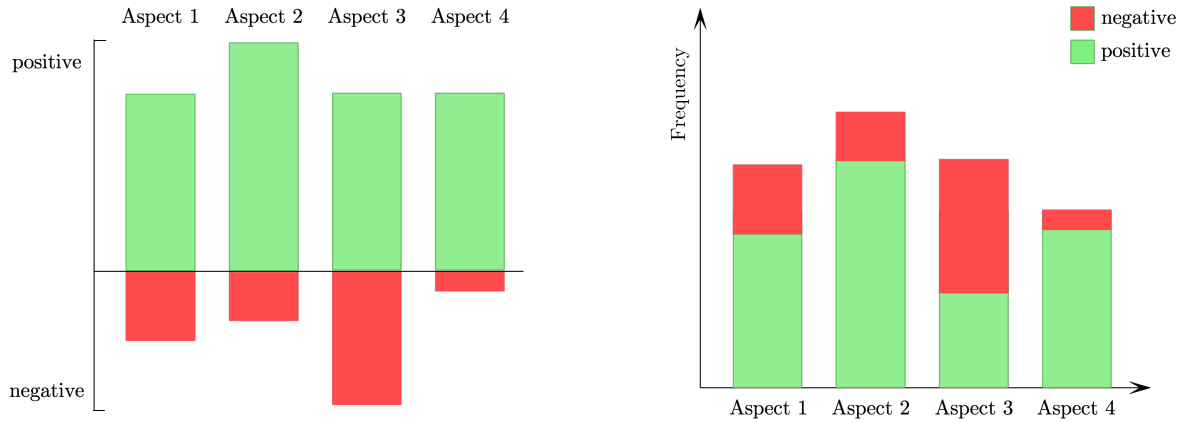
Furthermore, the user can select a cluster (in the figure, *"VW, service"* is selected) in order to view examples of sentences in each of the semantic categories, as shown in the lower part of the figure. This is a possible means to provide "evidence" from which the opinions are deduced, which are likely to prove helpful to the user [231].

Bounded summary statistics alone are typically not sufficient to provide the user with meaningful insight into the opinions contained in a document. Extreme percentages, for example, are less impactful in the context of small sample sizes — a statistic of 80% negative reviews has a much larger effect in the mind of the user if this is based on 1 000 reviews rather than on 5. Furthermore, merely indicating the average sentiment may produce equal results for a

unimodal[12] distribution around the mean, as for a bimodal distribution, with peaks at both the positive and negative end, for example, thereby discarding valuable information. Sentiment summaries therefore typically include the average sentiment score, the distribution of scores and the number of ratings associated with each score [231].

Unbounded summary statistics can be visually presented in various ways. As in Figure 4.10, for example, this can be done using size or extent. Another approach is to use common statistical visualisation tools such as histograms, which visualise frequency counts, and box plots, which are able to represent graphically the mean and the dispersion of observations simultaneously. Interesting variants of such plots have been employed for the purpose of visualising sentiment.

Liu *et al.* [177] proposed comparing products by visualising the number of positive and negative reviews in respect of each aspect of the product on opposite sides of a horizontal axis. A fictional example of this type of representation for one product is shown in Figure 4.11(a).



(a) *Shared zero line* (b) *Stacked representation*

FIGURE 4.11: *An illustration of the difference between representing frequencies in a histogram (a) on either side of the horizontal axis, as proposed by Liu et al. [177] and (b) stacked on top of one another, as it is typically done.*

This allows the user to compare positive and negative reviews separately and provides a much clearer representation than the traditional histogram, in which positive and negative review frequencies are simply stacked on top of one another, as shown in Figure 4.11(b) [231]. In the original paper, various products were compared in this manner by plotting several such graphs side-by-side and grouping the frequency bars by feature rather than by product, in order to allow the user to immediately determine which products excel at any given feature.

Gregory *et al.* [106] developed a compact representation of sentiments in reviews using *rose plots*, adapted to visualise similar features to traditional box plots. As shown in the example in Figure 4.12, *"petals"* are grouped into pairs representing opposites (*e.g. positive* and *negative* or *pleasure* and *pain*) and are assigned a certain direction in a circular graph. Positive and negative sentiments are, for example, depicted in the northern direction. The mean score of an attribute is represented by the solid line, and the colour bar depicts the *interquartile range*[13], just as in a traditional box plot. The user can then compare several feature pairs at a glance and likely remember the location of a feature of interest more easily than in a visualisation where features

[12] A unimodal function has only one local maximum or minimum [342].

[13] The first quartile, $Q1$, of a group of observations is the 25^{th} percentile, or the observation that is greater than 25% of the observations. The 50^{th} and 75^{th} percentiles represent $Q2$ and $Q3$, respectively. The *interquartile range* is given by $IQR = Q3 - Q1$.

are depicted sequentially [106, 231]. The documents shown in the figure, for example, exhibited largely positive sentiment alongside some negative sentiment, and often described their subjects as virtuous.

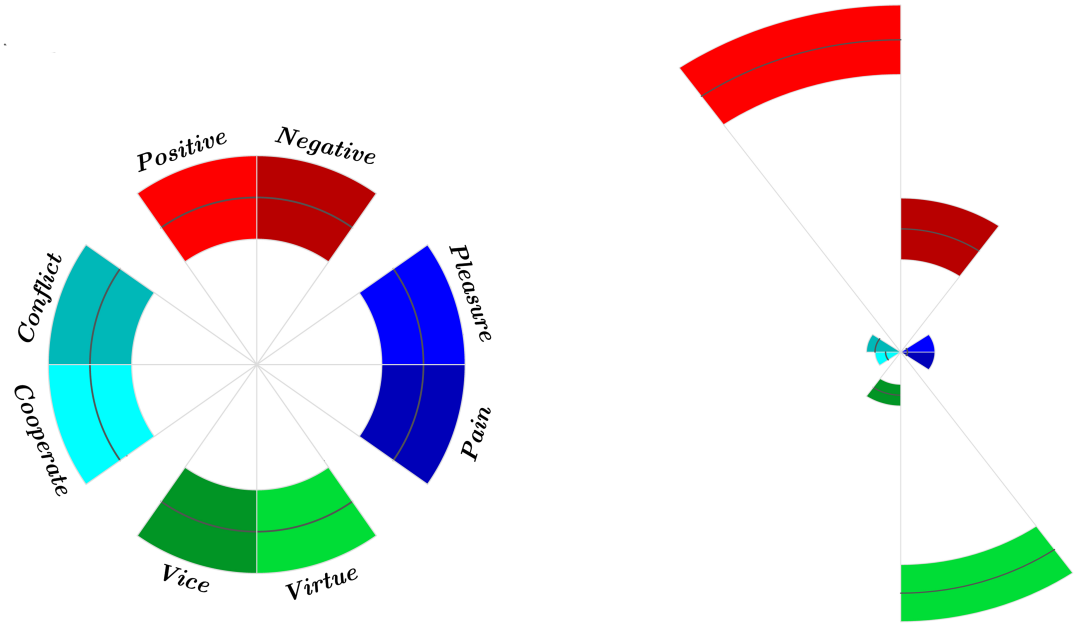


FIGURE 4.12: *A modified rose plot illustrating the scores associated with a group of documents in various affective categories (adapted from Gregory et al. [106]). The left-hand plot is the legend, indicating which categories appear in which direction. The right-hand plot shows the scores achieved by a group of documents in each of these categories, with the solid bar indicating the mean score, and the colour bar representing the dispersion.*

An alternative visualisation of the results of a sentiment analysis is shown in Figure 4.13. In this representation, developed by Moriniga *et al.* [202], products are plotted on the same system of axes as their *characteristic words*, whose presence in reviews about the product contributed significantly towards its predicted sentiment polarity. In this system of axes, which is created by applying PCA (described in §2.3.1), terms and products that occur close to one another are deemed closely associated. *Cellular phone A* in the figure, for example, is associated with what are considered positive terms, such as "*fast*" and "*no problem(s)*," whilst *Cellular phone C* is associated with negative terms, including "*slow*" and "*doesn't work*" [202, 231].

Finally, one can include several other dimensions of importance to offer more insight into the opinion content. Average sentiment may, for example, be plotted over time, as was done by Ku *et al.* [164]. Sentiment values can also be plotted as a *heat map* superimposed on a geographical map, where regions associated with positive sentiment are coloured differently from regions with negative sentiment, and where the shade of the colour varies with the strength of the sentiment. This approach was taken in a paper by Shook *et al.* [284].

4.3 Chapter summary

The first section of this chapter was devoted to a discussion on the processing of unstructured data. The chapter opened with a definition of data and the various types and forms they can assume, as well as an outline of the generic data science process. Subsequently, data processing steps related to unstructured text data were described, namely *tokenisation*, *stemming* and

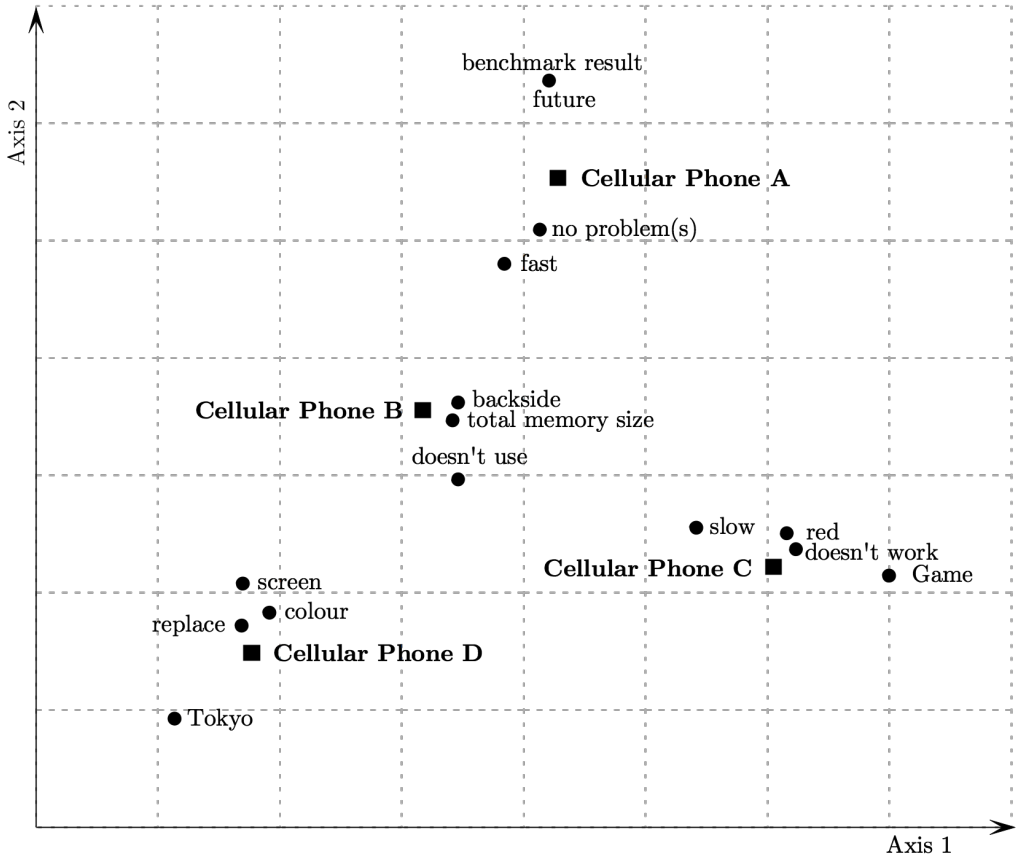


FIGURE 4.13: *Visualisation of products and their characteristic words in a two-dimensional space generated using PCA (adapted from Morinaga et al. [202]).*

lemmatisation (*normalisation*). On a related note, various types of *features* were reviewed, which may be used to represent an unstructured text in the form of a structured vector. These included *term-based*, *linguistic* and *topic-oriented* features. Furthermore, *dimensionality reduction* and *feature selection* techniques were discussed, including *dictionary size reduction*, *feature transformation* and *representation learning*.

Sentiment analysis of unstructured text data was considered in the second section of the chapter. A brief timeline of research in the field was first given, followed by a structured categorisation of the research into sentiment analysis *tasks* and *levels of granularity* pursued in the analysis. Subsequently, *approaches* to classifying document-level sentiment polarity were discussed at the hand of a popular taxonomy of techniques. During this process, the taxonomy was revised to reflect the current state of the literature. Finally, the subsequent step of *summarising* the results of the analysis was considered by reviewing *textual* and *visual* summaries of sentiment data.

CHAPTER 5

Decision support systems

Contents

DSSs are computerised systems designed to support complex decision making and problem solving through the use of data and models [244, 281]. According to foundational research by Alter [4], such systems possess the following three major characteristics:

(i) They are designed to *facilitate* a decision process,

(ii) they *support* rather than automate decision making, and

(iii) they are *responsive* to the changing needs of decision makers.

In this chapter, the notion of a DSS is described in more detail according to the major components these systems share before a distinction is made between various types of DSSs. Subsequently, popular methodologies for developing such systems are outlined, and this is followed by a discussion of methods for achieving quality assurance, system verification and validation in the context of DSS design.

5.1 Major components

There are a variety of different *generic architectures* which attempt to encapsulate the essential components of a DSS. One popular architecture describes a DSS as comprising a *language system* and a *presentation system*, which hold the set of messages that the DSS can accept and emit, respectively, as well as a *knowledge system* housing the knowledge a DSS has created and stored, and a *problem-processing system* used to recognise and solve problems during the decision-making process [127, 288].

Many researchers also consider a DSS as consisting of three major components, namely (1) a *database component*, which houses functions for managing and storing internal and external data, information and knowledge, (2) a *model component*, which provides users with access to a variety of models and assists in decision making, and (3) a *dialogue manager* or *user interface component*, which enables a user to interact with the system and obtain information by means of interactive queries, reporting and visualisation functions [244, 281, 296]. This is the view that is adopted in this dissertation. It should be evident that both views incorporate the same essential elements — the database component and model component closely resemble the knowledge system and problem-processing system of the first view, respectively, while the dialogue manager may be seen as the integration of the language system and presentation system.

A *communications component* is sometimes considered as a fourth essential element of a DSS. This component refers to the architecture and network of the DSS, and describes how the other components are integrated, as well as how the software and hardware are distributed on the system [244, 245]. These elements can, however, also be seen as inherent properties of the system, rather than one of its components.

5.1.1 The database component

The database component of a DSS may be described as a *database management system* (DBMS) that is responsible for creating, editing, deleting and maintaining a collection of data records [147, 218]. Kendall and Kendall [147] identify the following objectives that should be pursued in the construction of an effective database:

(i) Ensuring that data can be *shared* among users for a variety of applications,

(ii) maintaining data that are both *accurate* and *consistent*,

(iii) ensuring that all data required for current and future applications are *readily available*,

(iv) allowing the database to *evolve* as users' needs grow, and

(v) allowing users to construct their *own view* of the data without concern for the manner in which the data are physically stored.

A DBMS may be categorised into one of various types, based on its underlying *database model*, which determines how data are stored and managed. According to Obbayi [218], five such primary types may be distinguished, namely *flat file-based* databases, *hierarchical* databases, *network* databases, *relational* databases, and *object-orientated* databases.

A flat file-based database is perhaps the simplest type of database, in which data are stored in a human-readable text format or binary format, such as a `Microsoft Excel` spreadsheet or a

comma separated values (CSV) file. This type of database relies on the assumption that every data record consists of the same number of entries. Missing values are, therefore, not tolerated.

Hierarchical database models store data in a nested, tree-like format where *parent* nodes may have many *children*. This is suitable for storing data related to items exhibiting attributes and features, such as vehicles and their components and features, or books and their chapters and sections.

Network databases are similar to hierarchical databases, except that this model also allows for *many parent to many child* relationships. Although network databases have the advantage of flexibility, they are inefficient in practice, since searching for an item in such a database requires the computer to traverse the entire dataset. This type of model was popular in the 1960s and 1970s, but has since given way to the concept of a relational database [218].

Indeed, the most widely used database model is the relational database. In this model, data are stored in several meaningful tables, minimising the repetition of data and, consequently, also reducing the likelihood of errors and the required storage space. Table rows are referred to as data *records* and table columns as *attributes* of that record. Databases of this type are typically *normalised* to remove repeating groups and any dependencies between attributes. In the final normalised format, every record in a table is uniquely identified by a so-called *primary key*. Such keys are then used to logically link different tables together. The relationships between various tables are typically captured using an *entity relationship diagram* (ERD), which can accommodate *one-to-one*, *one-to-many*, *many-to-one* and *many-to-many* relationships [147]. Although this model may be less computationally efficient than others, this is typically not an obstacle in practice, given the processing power and memory available in modern computers [218].

The last, and most recently developed, database model is the object-oriented database. This type of database is closely related to, and designed to work in accordance with, object-oriented programming languages, thereby allowing applications to interpret the data as native code, effectively merging the data and the program. In this database structure, data are not stored in relational tables but are rather viewed as attributes of *objects* in the program. Objects, in the object-oriented paradigm, are people, places or things that are relevant to the system. Typical examples of objects are customers, products and orders. Displays or text areas in a *graphical user interface* (GUI) may also be objects [147]. Data in an object-oriented database are accessed using *pointers* (variables whose values are the addresses of other variables) [218].

As noted by French [88], there is no single type of database that always works well in the development of DSSs. Each database structure has its own strengths and limitations. A suitable DBMS should therefore be selected based on and, if necessary, tailored to the specific needs of the organisation for which the DSS is developed.

5.1.2 The model component

The model component of the DSS houses the models used to solve problems and formulate alternatives during the decision-making process. Such models can take various forms, including optimisation models, simulation models and models from the realm of *artificial intelligence* (AI) [244]. Some DSSs also make use of an *inference engine* — a special type of model used to process rules or identify relationships in data [245].

This component also includes some type of *model management system*, which manipulates and manages the models stored in the system, as well as the user interface [296]. An example of such a manipulation is the alteration of model parameters in order to allow the user to perform *'what-if'* analyses. Furthermore, when several models are used in combination, the model management

system is responsible for combining the model results. Often, multiple models are employed in a *competing* manner. This entails presenting a variety of models with the same data, evaluating the relative performances of these models and then accepting only the output of the best-performing model. Malhotra *et al.* [184], however, argue that models can also be used in a *complementary* manner. In their proposed framework, several models are used to classify a set of observations. If there is a certain level of agreement among the classifications generated by these models, the output is accepted. If, however, there is disagreement between the models, user input is elicited in order to reconcile the differences. The framework therefore works on a qualitative basis. It is also possible to combine the output of several models in a quantitative manner using en semble techniques, which were already introduced in §3.3.2 and §3.5.1. In this paradigm, model results are typically aggregated by computing the average (for regression problems) or by means of voting (for classification problems). The output of a model may also be weighted by a measure of its performance in respect of the training or validation data.

For some DSSs (specifically *model-driven* DSSs, which are introduced in a following section), the model component may be paritioned into three stages, namely *formulation*, *solution* and *analysis* [281]. During the formulation stage, a model is generated in a form acceptable to a model solver, which is used during the solution stage to algorithmically solve the model. Finally, the analysis stage entails conducting the aforementioned *'what-if'* analyses and interpreting model results.

5.1.3 The user interface

The interaction between a human and the computer (*human computer interaction*) is predominantly facilitated by a user interface. Since this interface often facilitates the only interaction a user has with a DSS, its design is essential to the success of the system.

There are four primary interface styles [245]. *Command-line interfaces* are the oldest form of interface, where a user issues typed commands to the computer. This form of interface can be powerful, since the user is given considerable freedom and control over the processes executed by the computer. The user must, however, be adequately trained and be familiar with the available commands of the system. *Menu interfaces* are slightly easier to use, since the user selects tasks or functions to be performed by the computer from a list of available options. Such an approach can become tedious, however, when complex tasks are performed, which require switching between several menus. GUIs are the most commonly used type of interface, where users have direct control of visible objects and can initiate actions *via point-and-click* mechanisms rather than *via* complex typed commands. Such an interface may furthermore be enriched with *multimedia* (graphics, audio and animation) and *hypermedia* (non-linear representations of multimedia that are connected *via* links, such as the *World Wide Web*). Finally, *question-and-answer* and *natural language* interfaces follow a dialogue-like structure between the user and the computer. This exchange typically takes place in the form of natural language. Consequently, such a system can easily be used by humans without training, but it requires additional processing functions at the computer's end. Such interfaces have historically rarely been used in DSS design [245], but are becoming increasingly popular with the rise of intelligent virtual assistants [61].

The goal of a user interface is an important factor in the design process. Spolsky [293] differentiates between system *learnability*, the ease with which a first-time-user can make use of an application, and *usability*, the ease-of-use for an experienced user. The former is an important objective, for example, when designing interfaces for guide systems in shopping malls or self-service systems in restaurants, cinemas or airports. In the design of systems intended for repeated use, however, usability is typically deemed more important.

Design considerations that influence the usability of a system include *intuitiveness*, *predictability* and *feedback*. Intuitiveness may be achieved by implementing a logical flow in the design, from left to right and top to bottom, and by maintaining consistency [147]. Information should, for example, be located in the same area each time a new display is accessed, similar items should be grouped together consistently and identical terminology should be used throughout the system [147, 245]. In order to further enhance a system's intuitiveness, information load should be reduced by making use of icons and graphic displays in favour of text, using simple language and limiting the displayed information to that which is necessary for the user's immediate requirements [245].

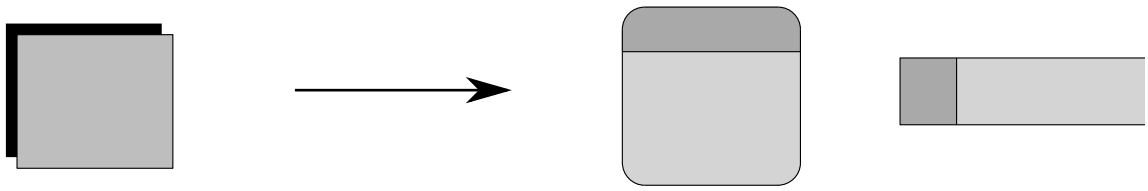
Predictability entails setting an accurate expectation of what can be done and what is about to happen using descriptive messages, labels and icons [126]. Sequences of actions should be partitioned into groups with a beginning, middle and end in order to create a sense of relief and accomplishment on the part of the user, and allow for preparation for the next sequence of actions. Furthermore, users should be given a sense of control by allowing actions to be reversed and by ensuring that no surprising behaviour by the system occurs [245].

Feedback, according to Hogue [126], echoes Newton's third law of motion — every action must have a visible, understandable and immediate reaction. If a user enters information or clicks a button, a message, progress bar or change in display should indicate what has happened. Warning messages and error messages should unambiguously state the nature and causes of any problem, as well as possible solutions. *Data validation* is an important process of the user interface, which ensures that no serious errors can be made by the user when entering invalid input. The use of drop-down lists and default entries are means of successfully reducing erroneous inputs in systems by minimising manual human data entry [147].

5.1.4 The communications component

The communications component of a DSS is primarily concerned with the system architecture. The *architecture* of an *information system* (IS) is a formal definition of its elements and subsystems. For a DSS, this architecture may be partitioned into several layers, each of which focuses on a different aspect of the system, such as the business processes that are completed within the system, the outputs and capabilities of the user interface, and the major software or hardware components [245].

The flow of information through a system may be represented visually using a *data flow diagram* (DFD). These diagrams represent the inputs and outputs of a system, as well as the processes that are completed within the system as a set of interrelated data flows [147]. Related business processes and the functions of the user interface may therefore be successfully documented using a DFD. Four symbols are typically used to create a DFD. These are shown in Figure 5.1.



(a) *Entity* (b) *Data flow* (c) *Process* (d) *Data store*

FIGURE 5.1: *The four symbols used in DFDs in Gane-Sarson notation [95].*

These symbols represent *external entities* that interact with the system, *processes* by which the system transforms inputs to outputs, *data flows*, which represent the directions along which data flow in the system, and *data stores*, which symbolise the storage of data. These elements are represented in so-called *Gane-Sarson* notation [95] in the figure. Another popular notation is the *Yourdon-Coad* notation [57].

Software and hardware considerations include how components of the DSS are physically connected and how software and data are distributed in the system. *Client-server architectures*, for example, attempt to balance the required processing power between a local computer (the *client*), which is typically responsible for presentation logic, and one or more *server devices*, which are responsible for data access logic and data storage. Other models include *client architectures*, *server architectures* and *cloud computing* [69]. Furthermore, the following factors are important considerations related to the software used when implementing a DSS [300]:

(i) *Licensing* describes whether the software is free or has to be purchased [79].

(ii) *Connectivity* refers to whether the DSS will operate online or offline.

(iii) The *language on server side* is the programming language used to develop the system, such as `C`, `Java`, `R` or `Python` [283].

(iv) The *integrated development environment* (IDE) is a software package that provides comprehensive support for writing, testing and debugging code [356].

(v) The *DBMS software* is used to store the data of the system, such as *MySQL* [291]. This is applicable when the database is extensive or complex.

(vi) The *application framework* is used to simplify the process of developing applications by providing libraries and templates aimed at promoting the reuse of code. If the final system is executed *via* the user's Internet browser (*i.e.* if it is *web-based*), the relevant framework is referred to as a *web application framework* [161].

During the development of the system, decisions have to be made with respect to each of the considerations above, in accordance with the needs that are to be met by the DSS, as well as the available resources.

5.2 Types of decision support systems

A taxonomy of DSSs was first proposed by Alter [4], which distinguished between seven types of DSS. Various other classification schemes have since been proposed based on criteria such as user type, *normativity*[1] or level of generality [288]. In this dissertation, the typology by Power [245] is adopted, which builds upon Alter's taxonomy and classifies DSSs according to the dominant technologies employed.

Data-driven DSSs place emphasis on enabling access to and manipulation of large collections of structured data. This class of DSS includes elementary tools facilitating query and retrieval of databases, *data warehouses*, which enable the manipulation of data integrated from multiple sources, as well as *online analytical processing* (OLAP) tools, designed to assist decision makers

[1] Normativity here refers to the type of support offered by the system. For example, a DSS may offer *passive assistance* through simple data analyses or *prescriptive support* by recommending actions or solutions [288].

in gaining insight into data through fast, interactive access to various transformed views of the raw input. In recent years, OLAP tools have often been replaced by *data mining* tools, which employ concepts from statistics and AI to elicit patterns and rules from data [281].

A *document-driven* DSS may be viewed as an extension of a data-driven DSS and derives its functionality from the storage, retrieval and processing of *unstructured* data such as documents, web pages and multimedia [288]. A search engine, for example, is associated with a document-driven DSS [245]. For both data-driven and document-driven DSSs, the data component is the most prominent element of the DSS architecture.

The focus of a *model-driven* DSSs is the access to and manipulation of a quantitative model, such as an optimisation model or a financial model. Data and user-specified parameters are employed in order to analyse a situation, but model-driven systems are typically not data intensive. The model or models are the dominant component in the DSS architecture. OLAP systems that enable complex analyses of data may be classified as hybrids of data-driven and model-driven DSSs with functionality for modelling, data retrieval and data summarisation.

Knowledge-driven DSSs, often also called *intelligent* DSSs or *suggestion* DSSs, recommend actions to decision makers based on *knowledge* that is stored in the system as a collection of facts, rules and procedures about a specific domain or problem [112]. The primary component of such a system is, therefore, the *knowledge base*, which is either viewed as a separate component next to the traditional model component, data component and user interface, or as an integral part of one or more of these components [320]. The process of *knowledge discovery*, employed to build such a repository of knowledge, is often conducted using an inference engine, which searches for hidden patterns and rules in a large database by employing data mining tools. In such a case, the DSS may be viewed as a hybrid data-driven and knowledge-driven DSS [245].

Finally, *communications-driven* or *group* DSSs are primarily aimed at supporting the exchange of data and information for collaborative decision making [288]. Such systems may be viewed as hybrid systems making use of communication technologies and decision models [244]. As such, a communications-driven DSS typically emphasises the system architecture and network design (the communications component).

5.3 Development methodologies

In order to develop a DSS, or any IS, a *systems development methodology* may be employed, which refers to the framework that is used to structure, plan and control the process of developing an IS [355]. These methodologies are often based on the *systems development life cycle* (SDLC), which identifies all activities required to build, launch and maintain an IS. There is little consensus in the literature on the contents of the SDLC, although most versions seem to incorporate a core set of processes or phases [271]. Kendall and Kendall [147] partition the SDLC into seven phases, as shown in Figure 5.2.

The phases of the SDLC may be implemented by adopting systems development methodologies in various ways, each differing in the manner in which each of the phases is planned and executed, and how the phases are combined [271]. Three popular development methodologies are the *waterfall* methodology, the *agile* methodology and the *object-oriented* methodology. The characteristics of these methodologies are briefly discussed in the remainder of this section.

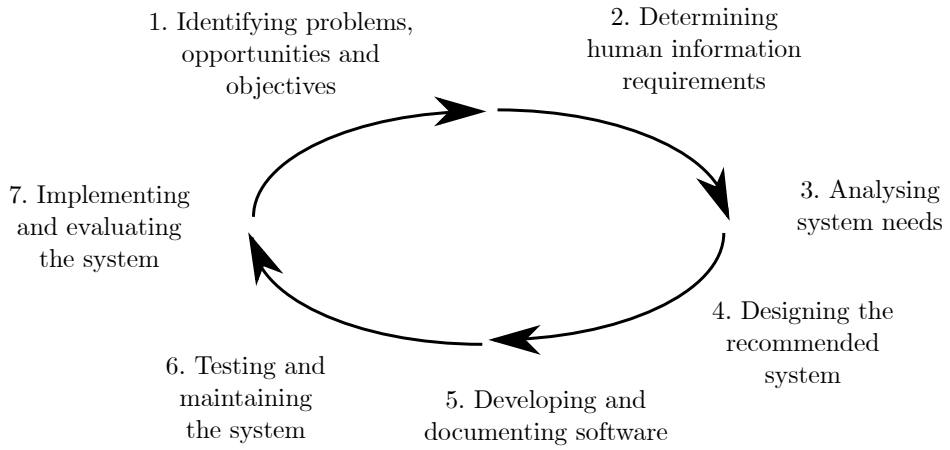


FIGURE 5.2: *The seven phases of the SDLC proposed by Kendall and Kendall [147].*

5.3.1 The waterfall methodology

In the waterfall methodology, each of the phases of the SDLC is completed sequentially[2]. The key results of each phase are presented and approval is sought from stakeholders before moving on to the next phase. Although backward movement between phases is not strictly disallowed in this methodology, it is difficult to do so. Typically, the process moves only in one direction, akin to the motion of water in a waterfall [69]. Advantages of this method include that it is thorough and systematic, and that the final system is well documented. All requirements are fully specified before any development begins, in an attempt to limit costly changes. This, however, also potentially renders the development process excessively long, and may result in a system that meets the original customer requirements, but has since become obsolete [69, 147]. This methodology is suitable when there is a sufficient amount of time available for system design and when the documentation of the system is important [147].

5.3.2 The agile methodology

The agile methodology was developed to address some of the shortcomings of the more traditional waterfall method by streamlining the SDLC process. Many of the documentation and modelling efforts are eliminated in favour of face-to-face communication with stakeholders and end users [69]. As such, many iterations are executed, in which the team developing the system produce prototypes of the system and demonstrate these to the stakeholders, who then give feedback to be incorporated during the next iteration. User requirements and system requirements are therefore elicited in an iterative manner. The advantage of this approach is that it is quick to implement and able to respond rapidly to changing needs. The resulting system is, however, typically badly documented, rendering maintenance more difficult. Agile development is suitable when time is of the essence and when stakeholders are satisfied with the incremental improvements associated with this methodology [147].

[2]Some researchers refer to the waterfall methodology simply as the *SDLC* or *SDLC approach* [147, 245].

5.3.3 The object-oriented methodology

The object-oriented methodology (OOM) may be characterised as a bottom-up approach to system development, in contrast to the top-down approach of the waterfall methodology [199]. According to this approach, the system is partitioned into *objects* containing data related to locations, events, people, or actual components of the system. As mentioned in §5.1.1, these objects may then be grouped into *classes* of objects that share similar characteristics. This partition is typically performed by employing the industry standard for documenting object-oriented systems, namely *unified modelling language* (UML) [147]. This methodology is similar to the waterfall methodology in that it is a structured, well-documented approach. OOM, furthermore, lends itself to the development of modular systems, which are easily partitioned into classes. Subsystems can then be developed individually, added to the system gradually and exchanged or updated separately from the larger system.

5.4 Quality assurance

The following three approaches to quality assurance may be followed during the development of a system, based on those proposed by Kendall and Kendall [147]:

(i) Design the system using a *top-down*, *modular* approach,

(ii) *document* the software using appropriate tools, and

(iii) *test* and *validate* the system.

In a top-down approach, the system is partitioned into subsystems and their requirements, or into logical *modules* that are *functionally cohesive*[3]. Such subsystems are easier to develop and *'debug,'* and are combined to form a modular, adaptable system. Furthermore, adopting this approach ensures that the overall objectives of the system are not lost in the design of detailed low-level components.

Documentation allows users, programmers and analysts to understand a system and its procedures without having to interact with it directly [147]. Consequently, it is easier to learn how to use the system, as well as perform maintenance and updates. Such documentation may include procedure manuals, documentation strings (commonly referred to as *'docstrings'*) that are attached to specific functions or objects within written code, as well as diagrams and models generated during the design process.

The final point is concerned with evaluating the final system by means of testing, or *verification*, and *validation*. These concepts are discussed in some detail in the remainder of this section.

5.4.1 Verification

Verification is concerned with building the model or system correctly [18]. This entails *'debugging'* the system in order to eliminate syntax and compiler errors, as well as logic errors. The former two errors are easily detected and resolved with the help of modern IDEs, which typically issue appropriate warnings. Errors in logic, however, are more difficult to detect. A structured approach is therefore necessary to ensure that the system is properly verified. An example of such an approach is shown in Figure 5.3.

[3] A *functionally cohesive* module is one in which all of the elements contribute to a single, well-defined task [301].

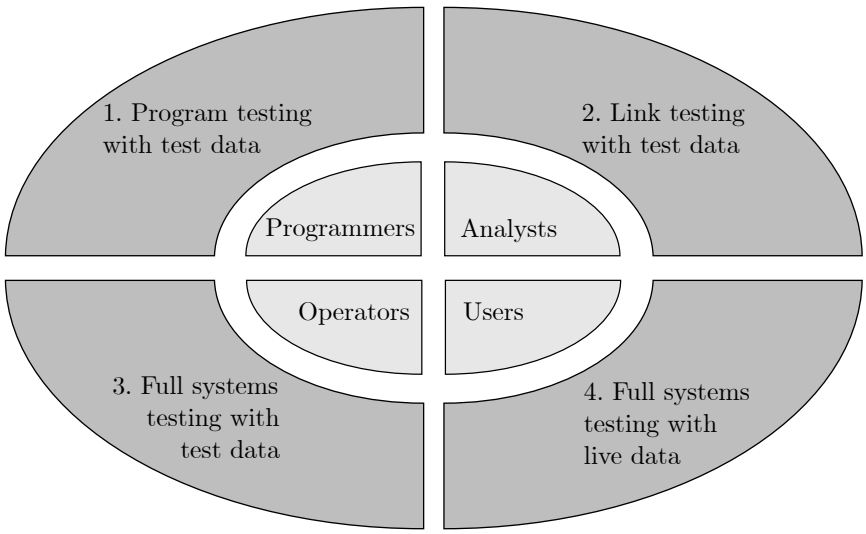


FIGURE 5.3: *A structured approach to system testing (adapted from Kendall and Kendall [147]). The elements of the inner circle represent entities involved in the testing phase, whilst the elements of the outer circle represent the four stages of the testing process.*

The first testing stage in Figure 5.3 is *program testing with test data*. During this stage, the authors of each module or program (the *programmers*) first *'desk check'* their work by following the executed logic on paper to ensure that it is executed correctly. Subsequently, artificial test data are generated and used as input into each module in order to verify that various scenarios are accommodated appropriately by the software. Any generated output is checked for errors and corrections are implemented.

During the second stage, *link testing with test data*, data flows between certain modules are tested to ensure that interdependent modules communicate correctly. This stage is executed using artificial test data representing typical cases as well as extreme, unusual or invalid values. *Analysts* are often involved in the preparation of these data.

During the third and fourth stages of the approach, *operators* and end *users* of the system become involved in the testing process. During *full systems testing with test data*, the system is verified as a whole using artificial data created explicitly to test whether the system objectives have been fulfilled. During this process, it is ascertained whether the documentation of the system is clear enough to understand by the user, whether the work flows facilitated by the system are efficiently executed and whether the output is correct and understandable.

Lastly, *full systems testing with live data* entails testing the entire system using real past data for which the output is known. The output of the system may then be compared to what is known to be correctly processed. Furthermore, the ease with which end users learn how to use the system is evaluated, along with their reaction to feedback generated by the system.

5.4.2 Validation

Validation is concerned with building the correct model or system [18]. This entails ensuring that the model possesses a sufficient level of accuracy within its domain [269]. Naylor and Finger [210] propose a three-step approach to validation that has been widely followed [18]:

1. Ensure that the model has high face validity,

2. validate model assumptions, and

3. compare the model input-output transformations to corresponding input-output transformations for the real system.

Although this approach was developed with reference to simulation modelling, the authors stipulate that it is not limited to this domain[4]. The first step of the approach entails ensuring that a model appears reasonable to users and other persons knowledgeable about the real system, typically referred to as *subject matter experts*. This also fortifies the *credibility* of the model, which refers to the level of confidence a user has in the model [269].

In the second step, critical assumptions made during model development are verified using statistical tests. If, for example, data are assumed to be distributed according to some theoretical statistical distribution, *goodness of fit* tests may be performed to test this hypothesis using samples of real data. In many cases, however, assumptions cannot be tested empirically. These assumptions may then either be abandoned in light of this difficulty, or retained as *'tentative'* postulates [210].

The final step of the approach entails testing the model's ability to predict the behaviour of the system under study, or evaluating how closely the output generated by the system resembles the expected or true output of the real system. In the realm of machine learning, this process is performed by training a model in respect of a portion of the available data and then testing or *validating* the model in respect of another portion of the available data, by evaluating its ability to reproduce the known output corresponding to these data in terms of relevant performance metrics, as described in §3.2.2 and §3.2.3.

5.5 Chapter summary

This chapter opened with a general description of DSSs according to their major components, namely the database component, the model component, the user interface and the communications component. Each of these components was discussed in turn and general guidelines or design considerations were given in each respective section. Subsequently, a distinction was made between various types of DSSs, namely data-driven, document-driven, model-driven, knowledge-driven and communications-driven DSSs. The development of DSSs, and ISs in general, was then discussed in terms of the SDLC and three popular systems development methodologies that are based upon it, namely the waterfall methodology, the agile methodology and the object-oriented methodology. Finally, quality assurance approaches were discussed, including the use of a top-down design approach and appropriate documentation, as well as verification and validation techniques.

[4]The term *model* may, therefore, also refer to a statistical or analytical model, or to an entire system.

Part II

Framework

CHAPTER 6

ECCO: A generic sentiment analysis framework

Contents

In this chapter, the sentiment analysis framework proposed in this dissertation is presented in some detail. An overview of similar existing frameworks from the literature is first given. This is followed by the description of a generic data science paradigm within which the proposed framework is set. Finally, a high-level overview of the proposed framework is given, and this is followed by a detailed description of its primary components using DFDs.

6.1 Similar existing frameworks

A number of frameworks for sentiment analysis have been proposed in the literature. In this section, an overview of existing frameworks is given, highlighting in particular those aspects which have not been adequately addressed by these frameworks.

In a notable paper, Khan *et al* [152] presented a generic sentiment analysis framework at the heart of which lies a *sentiment engine* that encompasses preprocessing and linguistic processing, as well as topic discovery and sentiment analysis tasks, as shown in Figure 6.1. This sentiment engine takes as input data from various sources and its results are presented *via* an online dashboard. At this high level of abstraction, the framework comprises the important elements required for a sentiment analysis framework: Input data are retrieved and preprocessed, information extraction is performed in order to describe the contents of the data, sentiment analysis is performed to classify the sentiment polarity of the data, and results are communicated in some summarised form through a GUI. It is, however, challenging to find existing frameworks in the literature that incorporate all of these elements at a lower level of abstraction (so that the framework may readily be applied), whilst retaining a generic nature.

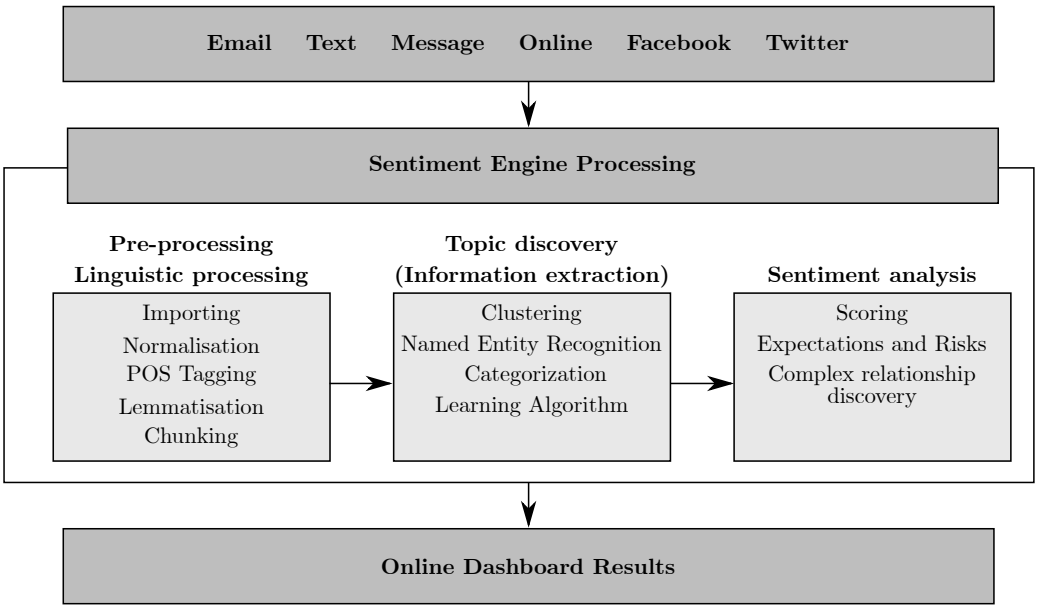


FIGURE 6.1: *A generic sentiment analysis framework proposed by Khan et al [152].*

In the remainder of their paper, Khan *et al.* [152] proposed a more specific framework, *Twitter Opinion Mining* (TOM) framework, in detail, which was tailored to the analysis of posts from the social media site *Twitter*. In contrast to the generic framework in Figure 6.1, the TOM framework encompasses only preprocessing and sentiment classification steps and exhibits a specific architecture with a lexicon-based hybrid classification model.

Several other frameworks [10, 44, 153, 155, 328] similarly propose a specific, rather than a generic, approach to sentiment classification, containing only data preprocessing and sentiment classification components. Yi *et al.* [353] also included aspect extraction in their framework, but did not describe a method for summarising the results according to the identified aspects. Typically, the aim in these frameworks is to improve classification performance for a specific domain or problem type [10, 44, 153, 155, 353] or a specific language [328] using preprocessing techniques and sentiment classification models specifically tailored to the relevant domain.

While an accurate sentiment classification model is necessary to evaluate opinionated content, it is not sufficient to form an understanding of the overall sentiment present in the data. An output such as "75% of customers are satisfied with the company" may give an indication of customer satisfaction levels, but such a result is not actionable. It is also necessary to identify, for example, which aspects of the company contributed to customer satisfaction or dissatisfaction as well as any trends that may indicate why certain customer segments are (dis)satisfied. Furthermore, it is not guaranteed that a model which is highly accurate in one domain will perform equally well in another domain. This is especially true for lexicon-based approaches, which are dominant in the literature related to sentiment analysis frameworks. Such methods often depend on lexical resources, which may not be available in a given domain or language, or may not yield the same level of performance when adapted to a new problem setting.

Perhaps the best-known opinion mining framework, the *Opinion Observer* published by Liu *et al.* [177], addressed the first of these issues. It focused on the summarisation of customer reviews in respect of competing products. Results were presented to the user in the form of a histogram, visualising the number of positive and negative reviews pertaining to automatically extracted product aspects for each competing product, as was already discussed in §4.2.4. This framework

also proposed the involvement of an analyst able to correct automatically extracted product features *via* a suitable user interface. The process of opinion polarity classification was not included in the framework, since the paper dealt with semi-structured customer reviews, where comments were already separated into *pros* (positive aspects) and *cons* (negative aspects).

Several other frameworks follow a similar approach to that of Liu *et al.* [177], grouping sentiment counts by aspects or product features, and either visualising these results or displaying excerpts from reviews in these aspect or product groups [11, 63, 71, 143]. Ren and Hong [258] generated similar visualisations according to topics, based on topic-based sentiment analysis using LDA. Some frameworks included additional steps in an attempt to render the output more informative for the user. De Lucena [67], for example, generated meaningful textual summaries which transformed the mentioned features and sentiment polarities into recommendations for action. Ordenes *et al.* [225], on the other hand, adopted a linguistic analytic approach to link the results of the classification to key components of the *value creating process*, namely *activities*, *resources*, and *context*.

Some frameworks go beyond this aspect-based summary to include other variables. Yussupova *et al.* [354] and Wu *et al.* [345], for example, included graphs visualising sentiment trends over time. Bank [17], furthermore, allowed the user to track certain customer satisfaction indices over time, as well as visualise common terms occuring in the corpus. Ganesan [96] incorporated powerful search functionalities that allow the user to retrieve sentiment summaries of only those reviews that match certain search criteria (*e.g.* reviews of hotels that are a certain distance away from a landmark and mention certain key phrases). Kasper and Vela [143], as well as Yaakub *et al.* [349], incorporated structured data on reviewers' demographics by displaying summary statistics of these demographics and providing summary reports for specific groups of customers. No frameworks were found, however, that explicitly facilitate the exploration of the relationship between *structured* variables and the content or opinion polarities of the *unstructured* data that are typically analysed. The only examples of frameworks facilitating a targeted analysis of model results are that of James *et al.* [136], who analysed which latent topics were the best predictors of the ratings of medical experiences by patients, and that of Yussupova *et al.* [354], who, similarly, analysed which features mentioned in reviews were indicative of the overall review sentiment using a decision tree analysis. In these examples, however, the relationship between features extracted from unstructured reviews were analysed, but they did not make use of any external data.

Others have integrated a sentiment analysis component into a larger social media analysis framework, which summarises additional information such as *likes*, *follows* and associated users or keywords for a particular account or search term over time [77, 227], or have made use of sentiment polarities to identify other factors, such as the nature of the relationship between a pair of users [352] or the underlying intent of a post [3, 115].

Only very few frameworks have attempted to address the problem of a lack of generalisability of frameworks. Typically, this has been done by designing frameworks in a modular, extendible manner so that they may be adapted to another problem setting or by incorporating several different models into the framework instead of a single, specific model.

Hsueh [128] proposed a generic software framework for opinion mining using object-oriented, modular constructs. Due to this design approach, the framework is flexible in respect of change and extension. The framework, however, provides little guidance on actually constructing the sentiment analysis models and does not include functionality for results summarisation or presentation. Lloret *et al.* [180] proposed a unified framework encompassing information retrieval, opinion classification and opinion summarisation. This framework is also modular in nature and can therefore be extended beyond the functions it was initially designed to facilitate (which

include only lexicon-based sentiment modelling approaches). El Sisi *et al.* [80] built upon the framework of Lloret *et al.* [180] by redesigning the modelling component to facilitate sentiment classification using several machine learning models and two different feature sets. Lloret *et al.* [180] mentioned that each component of their framework may be tuned by choosing among different parameters and options. No guidance is given in the framework, however, as to how such a tuning procedure should be conducted. Isah *et al.* [134] proposed the only framework that could be found which included a generic description of both machine learning and lexicon-based approaches to sentiment analysis. During the application of their framework, however, only one of the approaches was selected and no comparison was carried out of the relative performance of the different approaches. The hybrid classification framework by Liu and Lee [179], on the other hand, is capable of comparing the performance of three different modelling approaches, but does not include any reporting features for summarising the results of these models.

Schouten *et al.* [275] proposed the *Heracles* framework, which facilitates the flexible development and comparison of text mining algorithms (including sentiment classification algorithms) and incorporates various preprocessing activities as well as a structured comparison of machine learning algorithms. Important components of the sentiment modelling phase, such as feature engineering and hyperparameter tuning are, however, not addressed, since automated packages (*e.g. Weka*) are used for this purpose. Furthermore, the comparison of lexicon-based models and an analysis of model results are not facilitated.

In summary, the existing literature on frameworks for evaluating opinionated data is primarily concerned with providing frameworks for deploying single, specific and typically lexicon-based models by following predefined sequences of preprocessing activities and employing a predetermined set of features. Even in rare cases where a comparison of several models is facilitated, little guidance is provided within the framework in respect of model development. Furthermore, although research is certainly conducted on the use of machine learning models for sentiment analysis, as discussed in §4.2.3, the machine learning approach is rarely included in *holistic*[1] frameworks for sentiment analysis such as the one developed in this dissertation. According to Kumar *et al.* [165], the most time-consuming activity in any process that makes use of machine learning is selecting the machine learning algorithm, generating and selecting suitable features from the data and tuning the parameters and hyperparameters of the model. To the author's best knowledge, there are no frameworks in the literature for sentiment analysis or opinion mining that address this activity when machine learning models are incorporated. Moreover, existing frameworks do not explicitly facilitate an analysis of the relationship of the results of the sentiment analysis model with additional structured data that may be available to the user. Finally, currently available frameworks do not take into consideration the problem that the effectiveness of certain preprocessing activities and associated algorithm choices may be dependent on the data set in question.

One possible reason for these shortcomings in the literature is the objective of most of these frameworks in terms of producing fully automated sentiment analysis systems. In this dissertation, however, a decision support approach is taken, where the objective is not to *automate* the process of evaluating opinionated data, but to *facilitate* this process. The framework proposed in this dissertation is distinguished from those in the existing literature largely by the following characteristics:

(i) The framework is *interactive* with a focus on facilitating rather than automating the evaluation of opinionated data by a user.

[1] A *holistic* framework here refers to a framework that incorporates, in some way, all of the elements found in the generic framework of Figure 6.1 (preprocessing, information extraction or topic discovery, sentiment analysis and results presentation).

(ii) Rather than implementing a specific model into the framework, the user is guided through the *model development* process, with a particular focus on the machine learning approach, where algorithm selection, parameter and hyperparameter tuning as well as feature selection are required.

(iii) Instead of merely presenting model results, the framework facilitates an *exploratory analysis* of these results, including an investigation of the relationship between sentiment and data attributes from supplementary, structured data sources.

Furthermore, the framework is designed with the objectives of generalisability and flexibility, as is elucidated in the following sections, further supporting its applicability to various problem domains.

6.2 A generic data science paradigm

As mentioned in Chapter 1, the aim in this dissertation is to develop a framework for evaluating a collection of unstructured opinions. Such a framework should preferably be generic and adaptable to various problem settings. In order to achieve this objective, it is proposed that the framework function within a generic data science paradigm and that it be *modular* in nature. Any modules incorporated into the framework may thus be exchanged, modified or deleted in accordance with new findings in the literature and the objectives of the analysis in question without disrupting the correct functioning of other modules of the framework. Additional modules may also be added to substitute for existing modules of the framework. By basing the framework on a widely adopted data science paradigm, it is ensured that all stages necessary for extracting knowledge and insights from data[2] are included in the framework, thereby preserving its generic nature.

A high-level overview of the proposed generic data science paradigm is shown in Figure 6.2. It comprises three primary components, namely a GUI, which facilitates communication with the user, a database, in which relevant data are stored, and a central functional component, which is partitioned into three subcomponents, namely a *processing* component, a *modelling* component and an *analysis* component. This reflects the three primary components of a DSS described in §5.1 (the database component, the model component and the user interface), as well as the three stages into which the model base of a model-driven DSS are typically partitioned (the formulation stage, the solution stage and the analysis stage). In the following description of the paradigm, its similarities with the typical data science process described in §4.1 are illustrated.

Raw data are provided by the user *via* the GUI. The *data collection* step of the data science process in Figure 4.1 is therefore assumed to have already been completed by the user. These data are then processed by m_1 functionally coherent modules residing in the processing component, which may perform several tasks, including data cleaning, data transformations (of unstructured data into structured data, for example) and normalisation. This component thus incorporates the data processing and data cleaning steps of Figure 4.1.

Processed data are then used as input for the modelling component. The m_2 modules residing in this component perform tasks related to the *model building and evaluation* step in Figure 4.1. The nature of the models included in this component is determined by the objectives of the particular study and may include classification models, regression models and traditional forecasting models. One may, for example, construct a model based on historical data which is able

[2]This transformation of raw data to knowledge and insights corresponds to the definition of data science given in §4.1 [73].

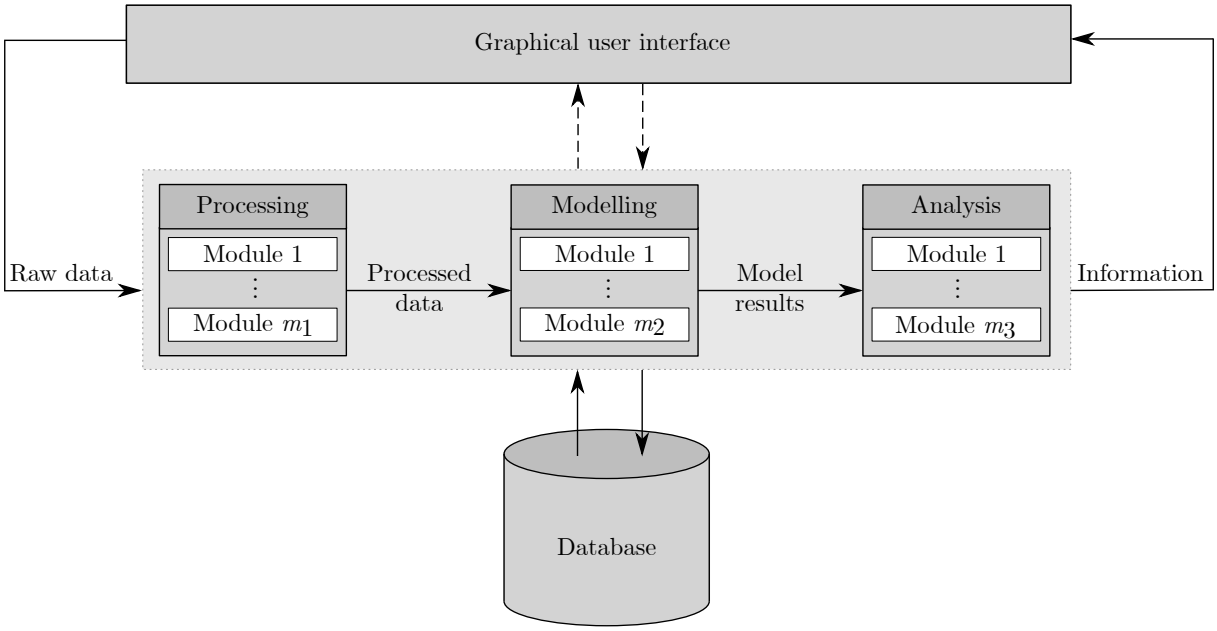


FIGURE 6.2: *A high-level abstraction of a generic data science paradigm.*

to classify observations into one of several categories. This model may then be applied to new data to obtain a classification for each new observation.

In order to extract valuable information and insight from these results, however, a subsequent analysis of the results is necessary. This may take the form of a simple visualisation of results or a more complex analysis aimed at identifying patterns explaining the model results. Such functions are performed by modules residing in the analysis component, as shown in Figure 6.2. The resulting information is then communicated to the user by means of the GUI, reflecting the *communication of results* step in Figure 4.1.

Throughout the execution of the modules in the processing, modelling and analysis components, data may be stored in and retrieved from a common database in order to reduce the inter-dependence between the modules in each of these components. The processed data may, for example, flow from the processing component to the database where they may be retrieved by the modules in the modelling component. The data flows in Figure 6.2 are shown to flow directly between model components merely for the sake of intuitive understanding. Furthermore, user input may be elicited and feedback given to the user at any stage of the process. The modules need not be executed sequentially, and can also be implemented in an iterative fashion, incorporating user input and allowing the user to alter this input after having observed the effects of changes to the input. In this manner, users may perform, to some extent, the *Exploratory Data Analysis* step in Figure 4.1. User interaction is, however, not required, as indicated by the dashed lines in Figure 6.2.

The approach depicted in Figure 6.2 also resonates well with general frameworks for *text mining* [3] [130, 217], in which *text refining* techniques are first applied to a collection of documents by preprocessing and then vectorising the text (resembling the processing component in Figure 6.2), and this is followed by a *knowledge distillation* process which includes predictive modelling, clustering and visualisation (resembling the modelling and analysis components in Figure 6.2).

[3] Text mining may be defined as the process of extracting hidden patterns or information from a collection of texts [32, 216].

6.3 The proposed sentiment analysis framework

Given the general paradigm in Figure 6.2, the proposed sentiment analysis framework can now be introduced. This framework is named ECCO (*Evaluating a Corpus Characterised by Opinion-bearing language*), alluding to the notion that the voices or opinions of the authors of the documents in the corpus *'echo'* throughout the analysis. A high-level overview of the framework is shown schematically in Figure 6.3, where the *generic* paradigm of §6.2 has been populated with the *domain-specific* modules proposed for evaluating opinionated content. This is reflected in the alteration of the name of the modelling component from *modelling* in general to *sentiment modelling* in particular. While the development of this framework takes place in the realm of sentiment analysis, the framework is not necessarily limited to this domain. The modules in the modelling component may also be configured to detect other aspects of unstructured text data, such as document type or topic. In the remainder of this dissertation, the focus will nevertheless remain on the modelling of sentiment.

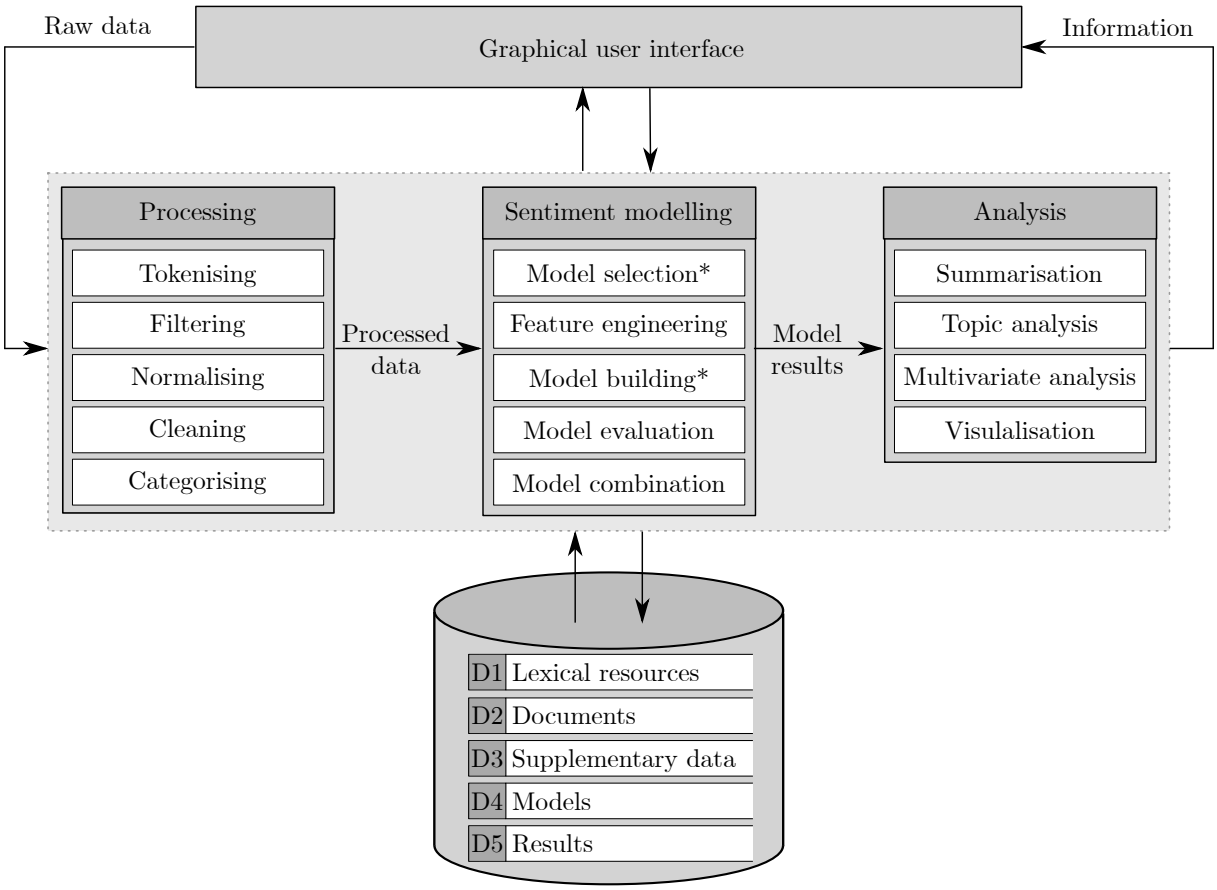


FIGURE 6.3: *A high-level overview of the proposed ECCO framework. Modules marked with an asterisk indicate summary modules which represent one module for the machine learning approach and one module for the lexicon-based approach to sentiment modelling.*

A seemingly small, but notable, difference between the ECCO framework in Figure 6.3 and the generic data science paradigm in Figure 6.2 is the transformation of the arrows between the GUI and the central processing component from dashed lines to solid lines. User input is therefore necessarily elicited in the ECCO framework during each of the processing stages. As mentioned in §4.2.3, the performance of a classification algorithm, particularly in the machine learning domain, depends highly on the model variant and feature set employed, as well as on

the data set on which the model is trained. Fully specifying the model and features employed in the framework would, therefore, require the delineation of various assumptions about the data sets for which the framework may be used. This is also true for some of the algorithms used to process the data and to analyse model results. The ECCO framework therefore *facilitates* the evaluation of unstructured, opinionated data by a user, rather than *automating* this process.

The only assumption that is imposed on the input data for the framework is that they are of an unstructured (free-form) text format and that a significant proportion of the data constitute sentiment-bearing content. These data, referred to as *documents*, may, furthermore, be enhanced with *supplementary data* which describe the document in more detail. These data, if present, are assumed to be presented in a relational form (*i.e.* stored in one or several tables which are linked to the documents and to one another by means of primary keys).

In the remainder of this section, each of the subcomponents of the ECCO framework are discussed in detail. Central to this discussion are the DFDs[4] of each of the subcomponents, which describe the processes involved in each subcomponent, as well as the data flows between these processes where required.

6.3.1 The processing component

The processing component comprises five modules[5] numbered 1.0 to 5.0 as shown in the *level-one* [6] DFD in Figure 6.4. As mentioned previously, the input data are assumed to be partitioned into two data sets — an unstructured component (the documents) and an optional structured component (the supplementary data). Modules 2.0–4.0 are designed to accommodate the unstructured component, whilst Module 5.0 is configured for the structured component. Module 1.0, on the other hand, is applied to both the structured and unstructured components of the data.

The first step in the processing component is the categorisation of data attributes. This entails specifying the location of certain data, such as the document text and their associated class labels, if present, as well as identifying the data types of attributes in the structured data component, so that subsequent modules can easily distinguish between qualitative and quantitative attributes. Furthermore, certain attributes may be designated as '*distinct*' attributes, indicating, for example, a location or date, that are treated differently in the analysis component later on.

The *raw* (unprocessed) documents are then *tokenised* or split into their constituent parts, as described in §4.1.1. The resulting sets of tokens are subsequently *filtered* in order to remove irrelevant or uninformative tokens from the data. Finally, the remaining tokens are *normalised* to their standard or root form in order to reduce redundancy in the data. These three processes are executed sequentially according to the desired settings selected by the user. These settings may include algorithms and parameter values to be used in any of the modules, and may also allow for the exclusion of certain processing steps. After the execution of the first three modules

[4]While the diagrams presented in this section are firmly rooted in the guidelines for generating DFDs from Kendall and Kendall [147], some of these standards have been adopted more loosely in order to avoid a cluttered presentation. If some data flows are, for example, required for a sequence of processes (*e.g. user input*), they are only explicitly shown to flow into the first process of such a sequence and are implicitly passed to subsequent processes. Any such changes should, however, not interfere with the intuitive understanding of the framework and are clarified in ambiguous cases.

[5]The terms *module* and *process* are used interchangeably in this dissertation in the context of DFDs.

[6]A DFD may be represented at various levels of abstraction. The level-zero diagram, also known as the *context diagram*, comprises a single process that represents the entire system. Each subsequent level represents the decomposition of parent processes into their child processes, where possible [147].

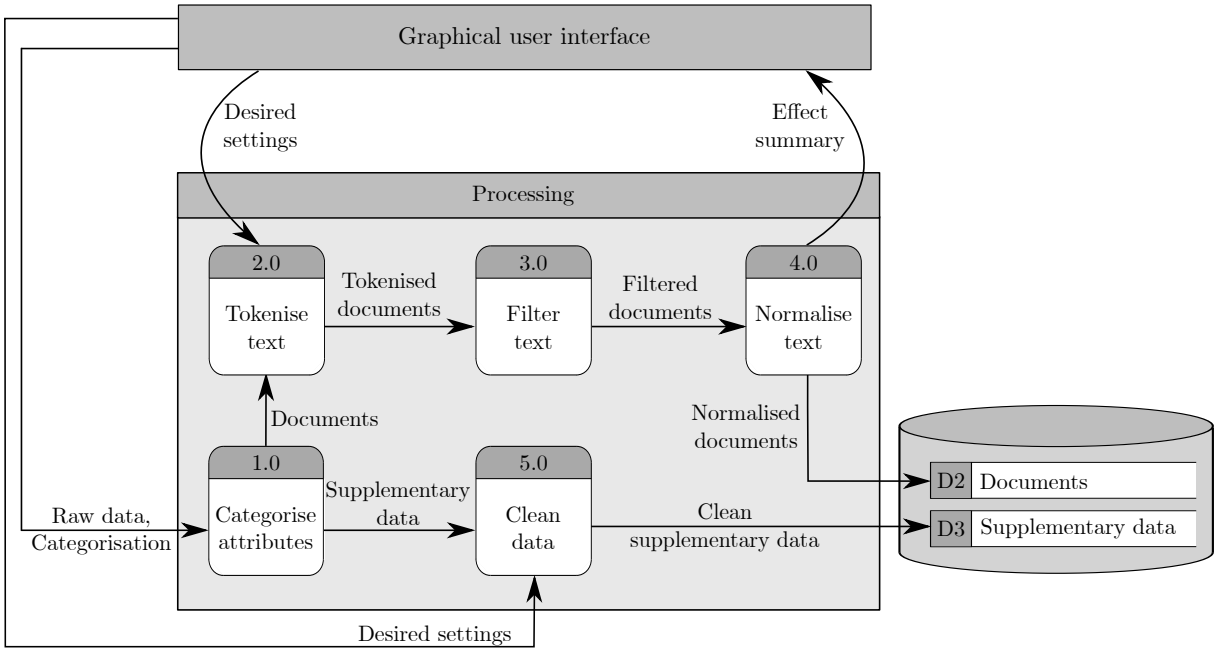


FIGURE 6.4: *Level-one DFD of the processing component.*

has been completed, the user is given feedback as to how these processing steps have affected the input data in the form of an *effect summary*. This summary should allow the user to gauge how effective the processing methods have been in transforming the data into a form that is useful for later analysis. Subsequently, the user is able to adjust the chosen settings and repeat the process until a satisfactory outcome is achieved. This iterative procedure is indicated by the curved arrows in Figure 6.4.

The structured supplementary data are subjected to a data cleaning process, the aim of which is to remove errors and inconsistencies in the data. Although this process could be executed in an iterative fashion similar to those in Modules 2.0–4.0, the processing of supplementary data is not central to the proposed framework, and it is therefore assumed that these data are extracted from a well-maintained database and that a single iteration of standard data cleaning techniques is sufficient.

The processed documents and supplementary data are then stored in a central database where they can be accessed by other modules of the framework. While any of the database types mentioned in §5.1.1 can be used in the ECCO framework without impacting its functionality, an object-oriented database is proposed. This type of database has the advantage that additional data, such as the categories of various attributes, can be stored in the same location as their associated tabulated or text data. Furthermore, functional modules and elements of the graphical user interface can be stored using the same object-oriented constructs and thus accessed using the same syntax.

A more detailed representation of the processing component is shown in its level-two DFD in Figure 6.5. Here the categorisation of data attributes (Module 1.0) is partitioned into three child processes 1.1–1.3. The first child process entails tagging distinct attributes that have a special function within the framework, as previously explained. This step also includes the distinction of documents from their class labels. Secondly, structured data attributes are categorised as qualitative or quantitative based on their data type. Lastly, the fields constituting the primary keys by which the supplementary data and the documents are linked are identified.

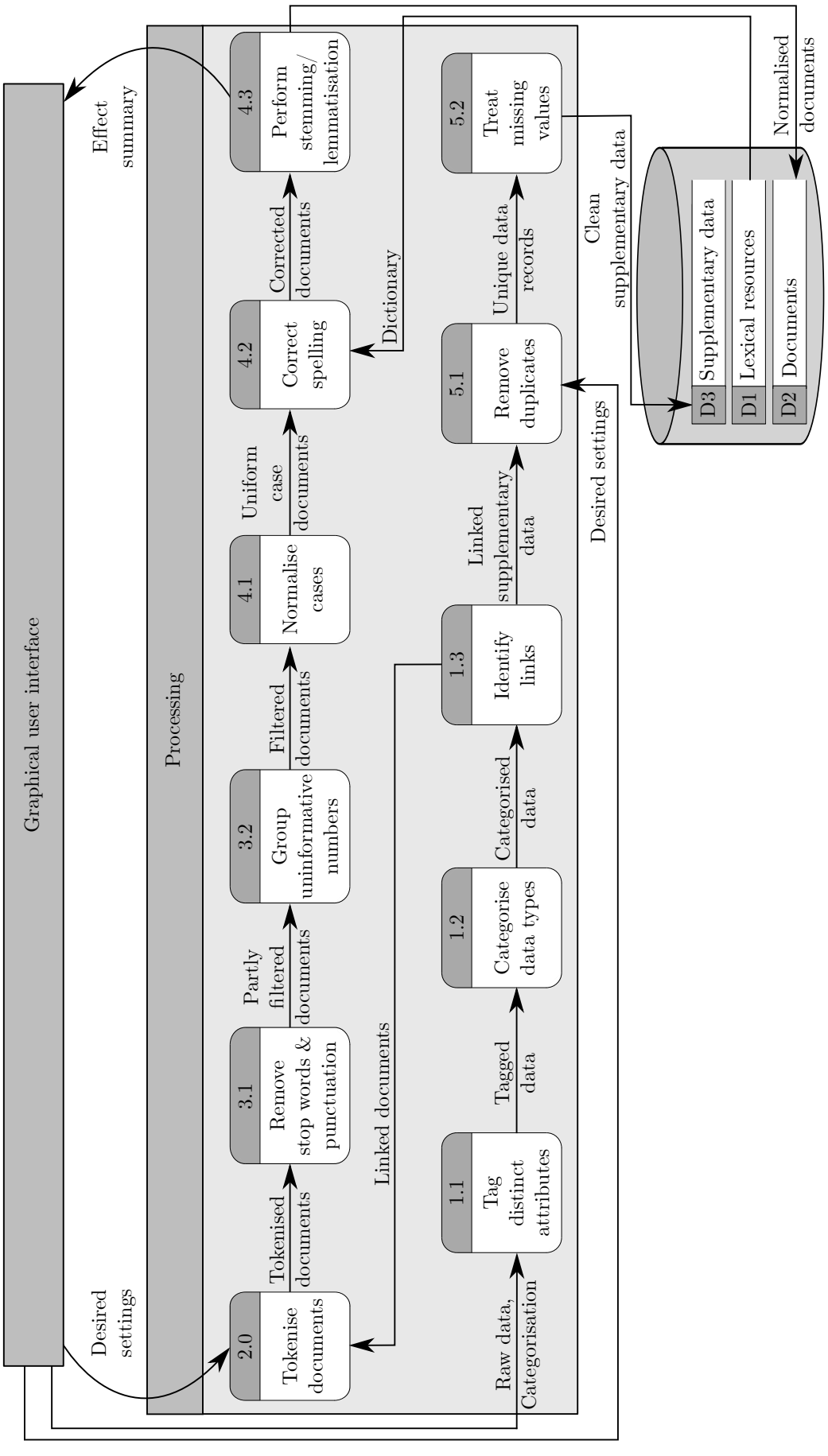


FIGURE 6.5: *Level-two DFD of the processing component.*

While it is possible to automate these processes to some degree, such an approach is not advocated in the proposed framework, since human intervention is still necessary to resolve ambiguous cases. An algorithm may, for example, automatically classify attributes with numerical values as quantitative. An incorrect classification would then be made, however, if the values are, in fact, enumerations of categories. Instead of indicating the departments of employees in a company by the department name, for example, each department could be represented by a number (*e.g.* 1 represents *accounting* and 2 represents *marketing*). In order to alleviate this problem, one might suggest implementing a rule that categorises numerical attributes as qualitative if the number of unique values observed for this attribute in the given data set is smaller than a certain threshold. This may, however, introduce further complications in cases where the data represent a narrow sample of a population. If the attribute indicates the number of years an employee has been working for a company, for example, a limited sample of new recruits could exhibit values ranging only from zero to three, again causing the algorithm to make an incorrect assessment. Eliciting user input in these cases allows for a simpler, quicker and more accurate completion of this process.

Module 2.0 is not exploded into its child processes, since it represents a simple process. Module 3.0, on the other hand, is partitioned into two child processes which apply three filters: One for uninformative words, one for uninformative punctuation and one for uninformative numbers. The removal of common stopwords was described as a technique for reducing the feature space in §4.1.3. This is typically achieved by comparing the tokens resulting from the tokenisation step with a pre-compiled list of stop words and removing these from the input data. Often, these lists also contain punctuation marks that are also typically removed from the data set. Most of these lists were, however, compiled for general text mining purposes and may therefore cause the exclusion of words important for sentiment classification, such as negation words (*e.g. 'not'* and *'no'*), intensifiers (*e.g. 'too'* and *'very'*) or punctuation indicative of sentiment (*e.g. '!!!'*). The user should, therefore, be able to customise such a list in order to exclude such cases. Lastly, uninformative numbers are removed from the data set. These include reviewer's phone numbers or amounts of currency, which are rarely informative, since any model would identify these as distinct features and therefore not be able to *learn from experience*. Such numbers may be grouped into a single feature indicating the presence of some numerical value, or may be removed entirely. In this case too, however, the user should be able to exclude certain numbers from the process. If the authors of the documents to be analysed were prompted to include, for example, a rating out of five or a monetary value as a gauge for their level of satisfaction, such numbers should be treated as separate features, since it can be assumed that several observations will exhibit these features.

The normalisation of documents (Module 4.0) is partitioned into three child processes. First, each of the tokens is converted to a common case (either all uppercase or lowercase letters). Then the tokens are checked for correct spelling and, where applicable, spelling correction is performed. This is typically done by comparing each word with entries in a standard dictionary and, if the word is not contained in the dictionary, successively removing, altering or exchanging letters in the word until it does match one of the dictionary terms. Two popular open source spell checkers are Hunspell [211] and Aspell [12]. Care has to be taken, however, when domain-specific words or jargon are encountered that are not contained in the dictionary, but are nevertheless spelt correctly. An investigation of word usage in the corpus to be analysed can provide guidance in such cases. After spelling correction has been performed, each of the tokens is transformed into a root form by means of a stemming or lemmatisation algorithm. The working of lemmatisation and popular stemming algorithms, such as in Porter's Stemming Algorithm and in the Lancaster Stemming Algorithm, were described in §4.1.1.

Finally, the data cleaning process in Module 5 is represented in the ECCO framework by two child processes, namely the removal of duplicate entries and the treatment of missing values. Typical data cleaning activities also include handling error values and outliers, as mentioned in §4.1. Without prior knowledge of the contents of the data, however, these activities are difficult to perform. A value of 1 000 kilograms for a person's weight, for example, surely constitutes an outlier value indicating an error that should be corrected. An unusually high credit balance may, however, be helpful to identify customers that are at risk or in need of assistance, and should not be reduced to a more common value. For this reason, these activities are excluded from the framework. After duplicate entries have been removed, missing entries are identified and treated according to the user's desired settings. Typically, this entails either imputing missing values with logical values or removing observations or attributes exhibiting a large proportion of missing entries.

6.3.2 The modelling component

The sentiment modelling component is shown schematically in Figure 6.6. It comprises seven modules, numbered 6.0–12.0 in continuation of the numbering of the processing component. Modules 6.0–8.0 are employed to develop machine learning models for sentiment analysis, whilst modules 9.0 and 10.0 facilitate the development of lexicon-based models. The rule-based approach, the last of the three approaches to sentiment modelling identified in §4.2.3, is not explicitly included in the ECCO framework, since this approach is typically used in combination with either machine learning or lexicon-based approaches and is therefore assumed to be included in the respective algorithms of these classes in such a case. A rule-based algorithm that does not make use of lexical resources or machine learning techniques can, however, easily be added to the modular framework. Modules 11.0 and 12.0 are aimed at evaluating and combining the models generated by the preceding modules.

The major feedback loop between the user and the functional modelling component is represented by the curved arrows flowing to and from the GUI in the figure, as before. The user selects a set of machine learning (abbreviated as ML in the figure) and lexicon-based algorithms to be configured from a given set. These algorithms may include any of the algorithms described in §4.2.3, such as Naïve Bayes, SVM or deep neural networks for a machine learning approach and Turney's [321] SO-A method for lexicon-based sentiment analysis. Furthermore, the user selects the desired settings employed to construct sets of features to be used in combination with the machine learning algorithms. The selected algorithms are then used to generate deployable models for classifying sentiment. For the machine learning algorithms, this entails finding good parameters by training the model in respect of a training data set and making adjustments to hyperparameters in an attempt to improve generalisability, as described in §3.2. Lexicon-based algorithms, on the other hand, require the creation of a suitable sentiment lexicon, as described in §4.2.3. The deployable models are then evaluated in respect of a test data set according to the user's evaluation criteria. Typically, these criteria take the form of one of the evaluation metrics described in §3.2.3, which may be applied to any classification model. Feedback on the models' performance is provided to the user by means of a *performance summary*, on the basis of which the user may change the input settings in an attempt to achieve the desired outcome more closely.

After having observed the performance comparison summary of the models, the user is also able to combine several models in order to form an ensemble, as described in §5.1.2. A similar performance summary of this ensemble is then relayed to the user. The model selection and

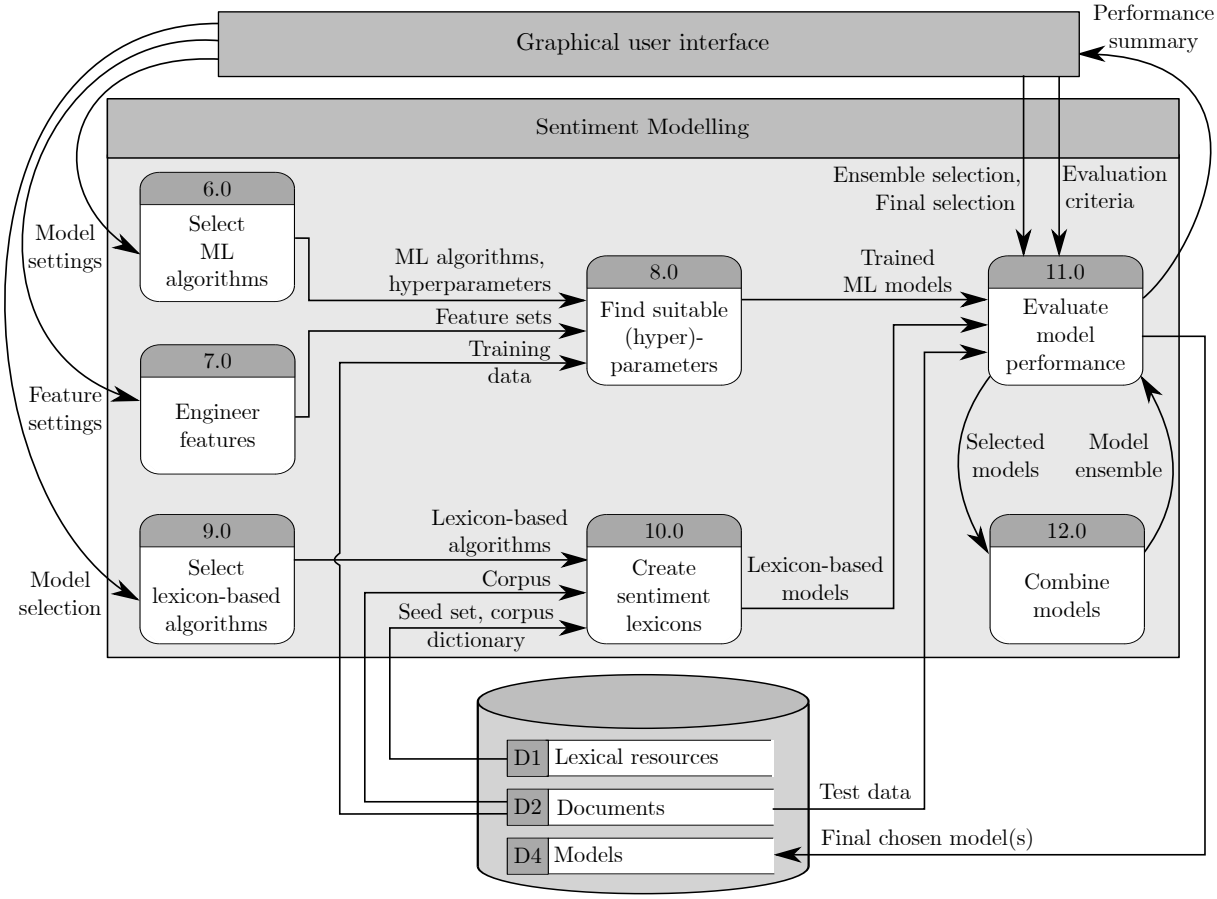


FIGURE 6.6: *Level-one DFD of the sentiment modelling component.*

ensemble selection process is repeated until a satisfactory level of performance is achieved. Then, the user selects the models that are to be stored in the central database for later use.

A more detailed representation of the modelling component is given by its level-two DFD in Figure 6.7. In this diagram, the process of feature engineering for machine learning (Module 7.0) is exploded into two child processes, namely feature generation and feature selection. Feature generation, in the context of sentiment analysis, refers to the process of text vectorisation. As described in §4.1.2, this may include extracting term-based features, linguistic features or topic-oriented features from free-form text in order to represent this text as a numerical vector. A bag-of-words model may, for example, be applied in conjunction with a specified range of n-grams to be extracted and an associated document model (*i.e.* the presence-based Bernoulli model, the frequency-based multinomial model or the weighted TF-IDF scheme). Feature selection refers to reducing the resulting feature space by reducing the size of the dictionary, applying feature transformations or training a model to learn a lower-dimensional representation, as described in §4.1.3. It should be noted that Modules 7.1 and 7.2 are used to *configure* feature generation and feature selection procedures, which are applied as part of the machine learning model during model training.

Process 8.0 is exploded into two child processes, aimed at estimating the parameters of the model and tuning the model hyperparameters, respectively. As is indicated by the curved arrows in the figure, these processes are executed in an iterative fashion. For each of the algorithm-feature combinations specified by the user, the model parameters are estimated for a fixed set of hyperparameters in respect of a set of training data. The model performance is then evaluated in

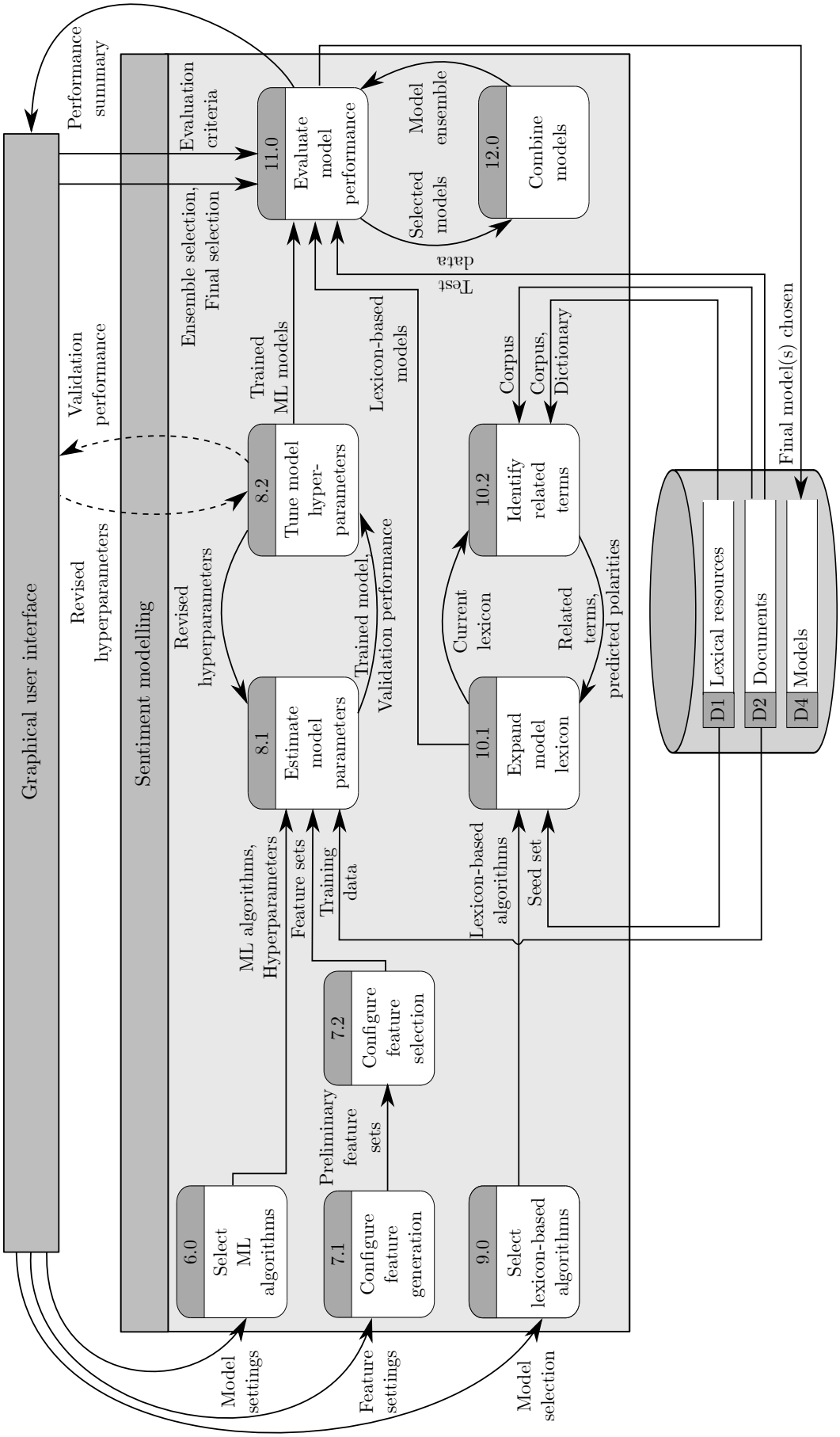


FIGURE 6.7: *Level-two DFD of the sentiment modelling component.*

respect of a validation data set (a certain proportion of the training data not used for parameter tuning), and the values of the hyperparameters are adjusted accordingly.

The algorithms employed in Process 8.1 typically include optimisation techniques, which are described in §2.4, such as gradient descent or the BFGS algorithm, and constitute hyperparameters of a model. Other hyperparameters include, for example, the number of nearest neighbours k considered in the kNN model, the tuning parameter C for SVM and the number of hidden layers included in an ANN. The hyperparameter tuning processes in Module 8.2 may include a manual search, a grid search, a random search or an *'intelligent'* search (using, for example, metaheuristics), as described in §3.2.1. If a manual search is employed, user input is elicited during every iteration, as indicated by the dashed arrows flowing between Module 8.2 and the GUI. If, however, a grid or random search is performed, a set or range of hyperparameter values to explore is specified by the user during algorithm selection in Module 6.0. Finally, if an intelligent search is carried out, user input is rarely required. A desired level of automation may therefore be achieved using the ECCO framework.

Finally, Process 10.0 is exploded into two child processes used to create sentiment lexicons. The lexicon-based algorithms selected in Process 9.0 are combined with an initial *seed set* of sentiment terms and their associated polarities. This set, which constitutes the initial sentiment lexicon, is then used to identify related terms and estimate their predicted polarities. This may be achieved by employing a dictionary-based method or a corpus-based method, as described in §4.2.3. If a dictionary-based approach is used, the seed set is expanded by its synonyms (which are assigned the same sentiment polarity) and antonyms (which are assigned the opposite sentiment polarity). These related terms are then integrated into the existing sentiment lexicon and the procedure is repeated until no other synonyms and antonyms are found. If a corpus-based approach is adopted, related terms are extracted *via* statistical co-occurrence patterns of the seed words with other terms in a given corpus. This corpus may either be generic, or may be supplied by the user and thus constitute a domain-specific corpus. In the latter case, the same documents used for training machine learning models are employed. Otherwise, the use of lexicon-based algorithms in the ECCO framework presupposes the availability of suitable lexical resources that are stored in the database prior to the analysis. The final lexicon-based model then includes the generated sentiment lexicon and the algorithm or rules employed to predict the polarity of a document using this lexicon (*e.g.* comparing the number of positive and negative adjectives from the lexicon that appear in the document).

As mentioned in §6.1, developing a suitable model is one of the most time-consuming activities in the machine learning process [165]. This activity involves three tasks, namely feature engineering, algorithm selection and (hyper)parameter tuning. Kumar *et al.* [165] refer to this as the *model selection triple* (MST) and further define the iterative procedure used to select a suitable MST as comprising three steps. The first is *steering*, where the user decides on one MST and specifies it. The second step is *execution* and entails the building and evaluation of the model by a computerised system. Finally, the user assesses these results in the *consumption* step and decides whether the process will be terminated or whether another MST will be tested during the next iteration. In a typical system, only one MST can be tested per iteration. Due to the importance of this process and its expensive nature, this leaves considerable room for improvement. In the ECCO framework, several efforts are made to address this problem.

In modules 6.0 and 7.0, the user is afforded the ability to specify several machine learning algorithms and several different settings for feature engineering in order to test various combinations during a single iteration. The framework thereby attempts to improve the efficiency of the steering step of the model selection process. Modules 8.1 and 8.2 facilitate the parameter tuning process. As mentioned earlier, this process may be automated, while still taking the

expertise of the user into account, if a random or grid search is employed in Module 8.2, and can further be improved by means of intelligent hyperparameter search algorithms. By transferring the computational load from the user to the computer, the execution step of the model selection process is simplified. Furthermore, allowing the user to compare several MSTs simultaneously can successfully guide the steering step for the next iteration. Finally, the consumption step is also enhanced by issuing the user with a meaningful performance summary of previously tested models for easy and rapid comparison purposes.

6.3.3 The analysis component

The working of the final component of the ECCO framework, the analysis component, is illustrated in Figure 6.8. It comprises six modules, numbered 13.0–18.0, that are designed to facilitate the analysis of the results returned by the sentiment classification model selected by the user. This component thus aims to execute and enrich the process of sentiment summarisation (§4.2.4). Module 13.0 facilitates the deployment of the selected model, taking as input the unlabelled documents that have been processed by modules 1.0–5.0 and the model that was developed in modules 6.0–12.0, and returning the sentiment class assigned to each document by the model. These *results* are stored in a local database for later use by other modules in the analysis component. Modules 15.0 and 16.0 are aimed at analysing the unstructured, textual component of the data, whilst modules 17.0 and 18.0 investigate the relationship between the structured, supplementary data attributes and the documents' sentiment classification. Finally, Module 14.0 enables the user to filter the results of the analyses in modules 16.0–18.0 according to the outputs of Module 15.0[7].

In Module 15.0, the corpus is analysed in terms of its underlying topics or frequently occurring key words. As is shown in the level-two DFD in Figure 6.8, this entails first performing topic analysis and then visualising the sentiment exhibited for each of the identified topics. As discussed in §4.1.2, topics can be extracted using dependency trees, noun phrase extraction or topic modelling techniques such as LDA (described in §2.3.3). In the latter case, model hyperparameters may have to be specified by the user. Additionally, topics may also be manually identified by the user. The visualisation of selected sentiment-topic relationships may take the form of a histogram, as in Figure 4.11, or alternative representations such as the rose plots of Figure 4.12 or low-dimensional representations of word distributions of Figure 4.13.

Module 16 is responsible for generating the summaries typically associated with text mining for the current selection of data. As discussed in §4.2.4, such traditional summaries take the form of textual summaries, which are generated using either template instantiation or passage extraction, or visual summaries, either of which is typically accompanied by sample documents for a clearer overview. These summaries may also make use of the topics discovered by Module 15.0 in order to give a more detailed overview.

Modules 15.0 and 16.0 provide the user with insight into the typical contents of the documents in the corpus and the possible underlying topic distribution, as well as an overview of the sentiment distribution in the entire corpus and for each identified topic. If supplementary data are available, it is also desirable to analyse the effect of these additional attributes on the sentiment distribution in order to further deepen the insight gained in respect of the corpus.

[7]Note that the execution of modules 16.0–18.0 is triggered by every change in the data contained in data store D5, as indicated by the dotted lines. Whilst this may not be traditionally accepted in DFD design [147], it is practically possible to implement such a configuration in the paradigm of *reactive programming* [15].

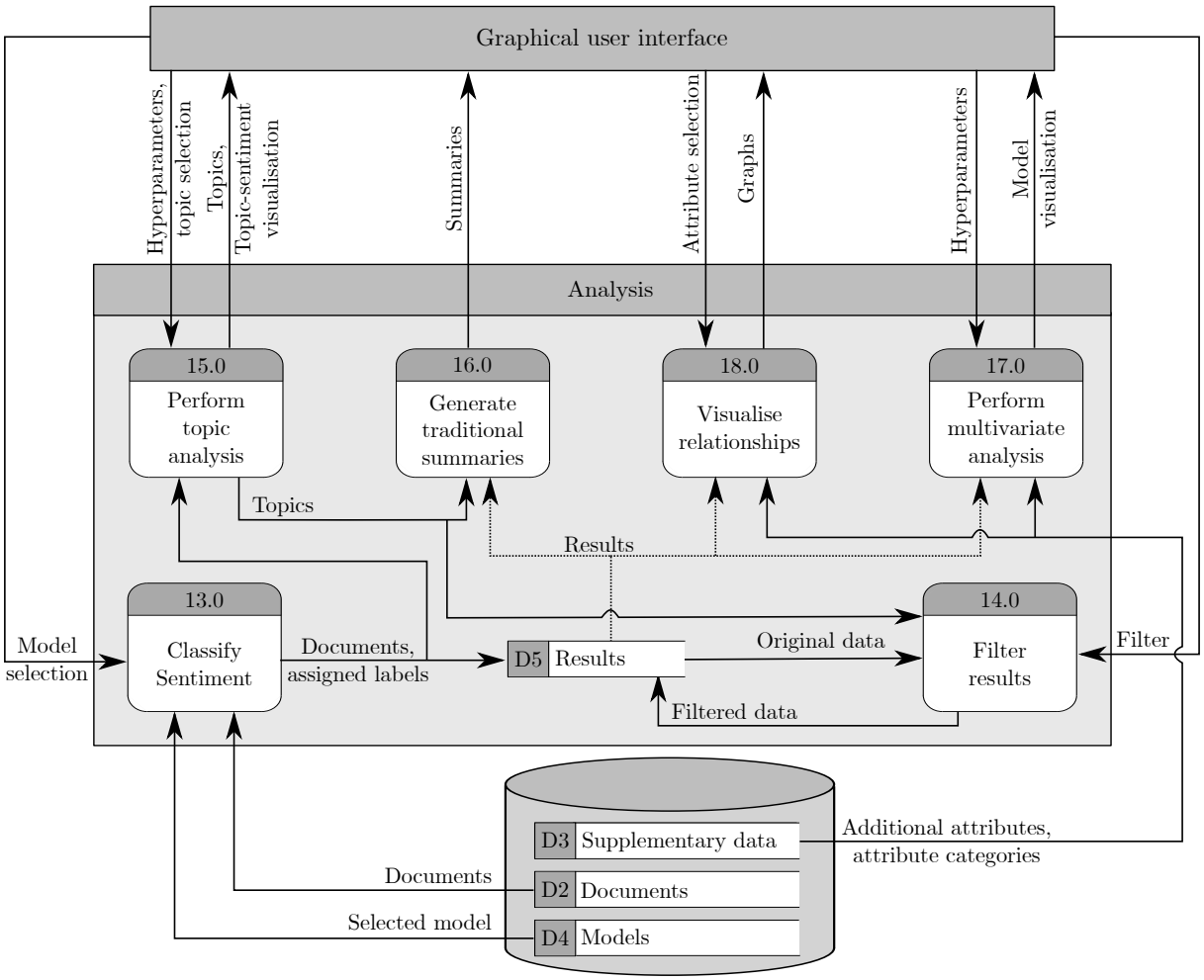


FIGURE 6.8: *Level-one DFD of the analysis component.*

This analysis is facilitated by Module 17.0 in the form of visual summaries. The user is afforded the ability to select a subset of attributes of the supplementary data, and a visualisation is then returned, depicting the relationship between these attributes and the relevant sentiment categories. As is shown in the exploded view in Figure 6.9, the attributes are first classified by their attribute type, which was categorised by the user during the processing procedure in Module 1.0. This allows for the creation of *type-specific* visualisations, such as histograms for qualitative attributes, box plots or scatter plots for quantitative variables, and geographical maps for location data. By exploring the visualsations generated for various attributes, the user may identify trends or relationships that may exist in the data.

In Module 18.0, a different approach is taken in the form of a *multivariate analysis*. More specifically, the approach proposed in the ECCO framework is to fit a statistical model to the data which seeks to model the relationship between the supplementary data attributes and the sentiment categories. Many such models may subsequently be analysed or visualised in order to reveal the *'rules'* or hidden relationships that have been discovered by the model. A classification tree (see §3.3.2) may, for example, be visualised by means of an hierarchical tree diagram. Similarly, the most *predictive* features and the most likely sentiment classification arising from various values of these features may be identified upon examining a trained logistic

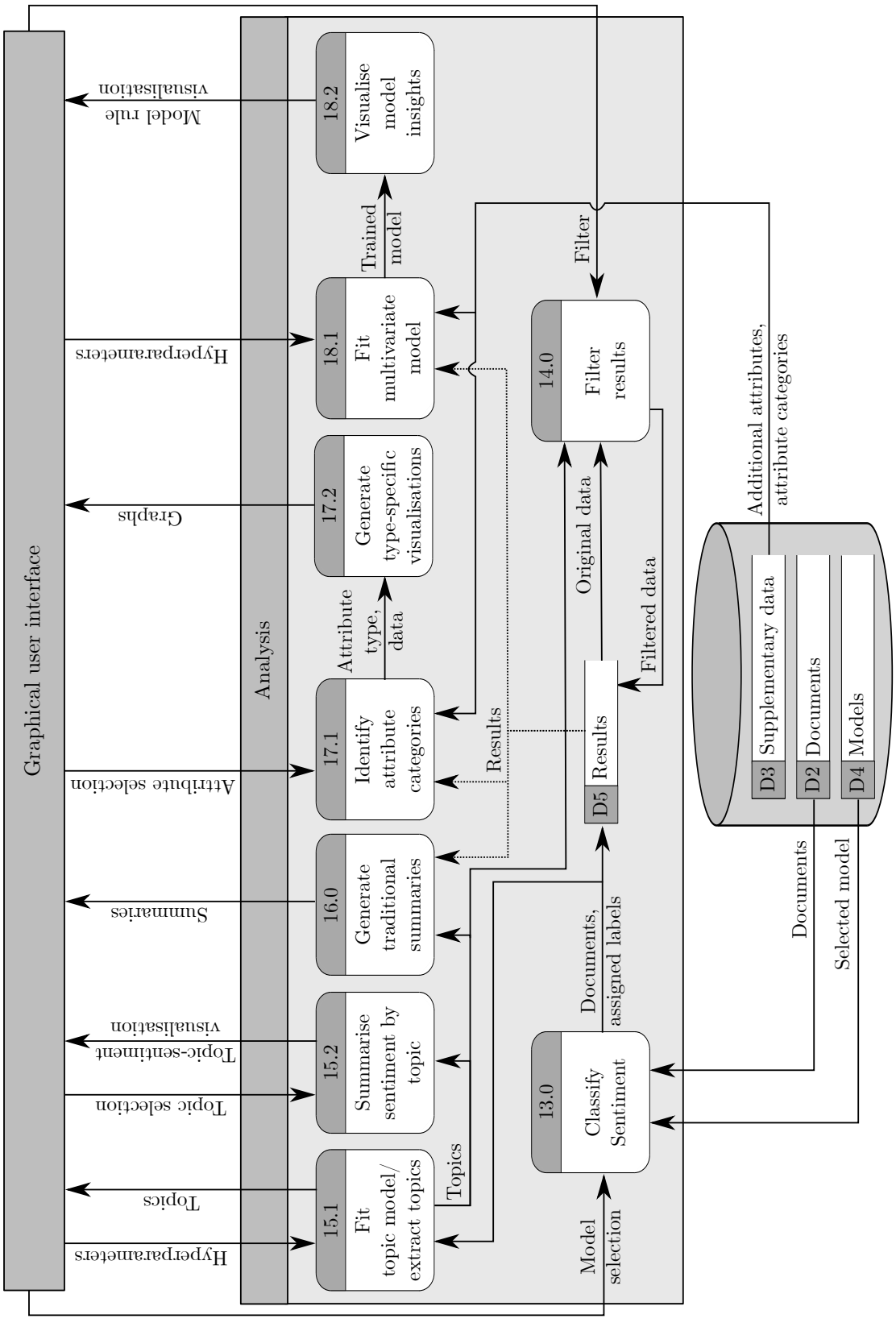


FIGURE 6.9: *Level-two DFD of the analysis component.*

regression model (see §3.3.5) or maximum entropy model (see §3.3.6). Adopting this approach, the user may discover hidden insights in the data that were not detectable *via* the visualisations in Module 17.0. As shown in Figure 6.9, the user may be required to specify hyperparameter values for these statistical models. As with most processes in the ECCO framework, these values may be adjusted iteratively until a desired level of performance or granularity is achieved by the model.

By executing modules 14.0–17.0 concurrently and iteratively varying the input parameters to these modules, the user is afforded the opportunity to explore the corpus with respect to the contents of the documents, the overall sentiment distribution and the relationship between sentiment and content, as well as the relationship between additional structured data and sentiment. In this manner, the three components of the ECCO framework facilitate the extraction of actionable insight from raw data.

6.4 Chapter summary

In this chapter, the framework proposed in this dissertation, called the ECCO framework, was introduced. The chapter opened with an overview of similar existing frameworks in the literature and highlighted the shortcomings of these frameworks, specifically with respect to a lack of generalisability due to full automation, the focus on the deployment of a single specific model rather than the facilitation of custom model development, and the lack of flexibility in analysing model results in order to extract actionable insights. Subsequently, a generic paradigm for data scientific analysis was proposed within which the ECCO framework itself was developed. Finally, the ECCO framework was described. A high-level overview of the framework was first given, before each of its three primary functional components, namely the processing component, the sentiment modelling component and the analysis component, were described in detail and illustrated by means of DFDs.

CHAPTER 7

Proof of concept implementation

Contents

In order to demonstrate the potential of the ECCO framework presented in the previous chapter, an instantiation of the framework was implemented in the form of a computer program. Generic components of the framework were populated with specific algorithms and elements in order to illustrate the functionality of the framework. This implementation, henceforth referred to as the *ECCO system*, is presented in detail in this chapter. First, however, an overview is given of the system's development and the design choices made within this context. Subsequently, the system is described in detail in terms of the three subcomponents of the ECCO framework, namely the processing component, the modelling component and the analysis component. During the description of each of these components, an account is given of the specific algorithms or processes chosen to populate that respective component of the framework. The chapter closes with a discussion on the verification and validation procedures followed during the development of the system.

7.1 System development

As mentioned in the previous chapter, the ECCO framework was designed to be modular in nature. Therefore, the object-oriented system development methodology (described in §5.3.3) was deemed suitable for implementing the instantiation of the framework. In particular, the system was partitioned into several *objects* and each of the modules contained within the ECCO framework was implemented using *methods* of certain objects, whilst the data in the database of the ECCO framework were stored as *attributes* of these objects. An illustration of this implementation architecture is shown in the *class diagram* in Figure 7.1. Blocks in the diagram represent classes, with class attributes shown in the top section and class methods shown in the bottom section of each class block. Attributes and methods marked with a '+' may be

accessed by other classes, whilst those marked with a '−' are only used by the class itself. Solid lines connecting classes indicate an *association* between classes, whilst lines with a diamond on one end indicate *whole/part relationships* [147]. If the diamond is not filled in, the relationship describes *aggregation*, in which case a class is composed of other classes, but these classes may also exist on their own. Finally, dashed arrows indicate that a class *realises* the operations offered by an interface [5].

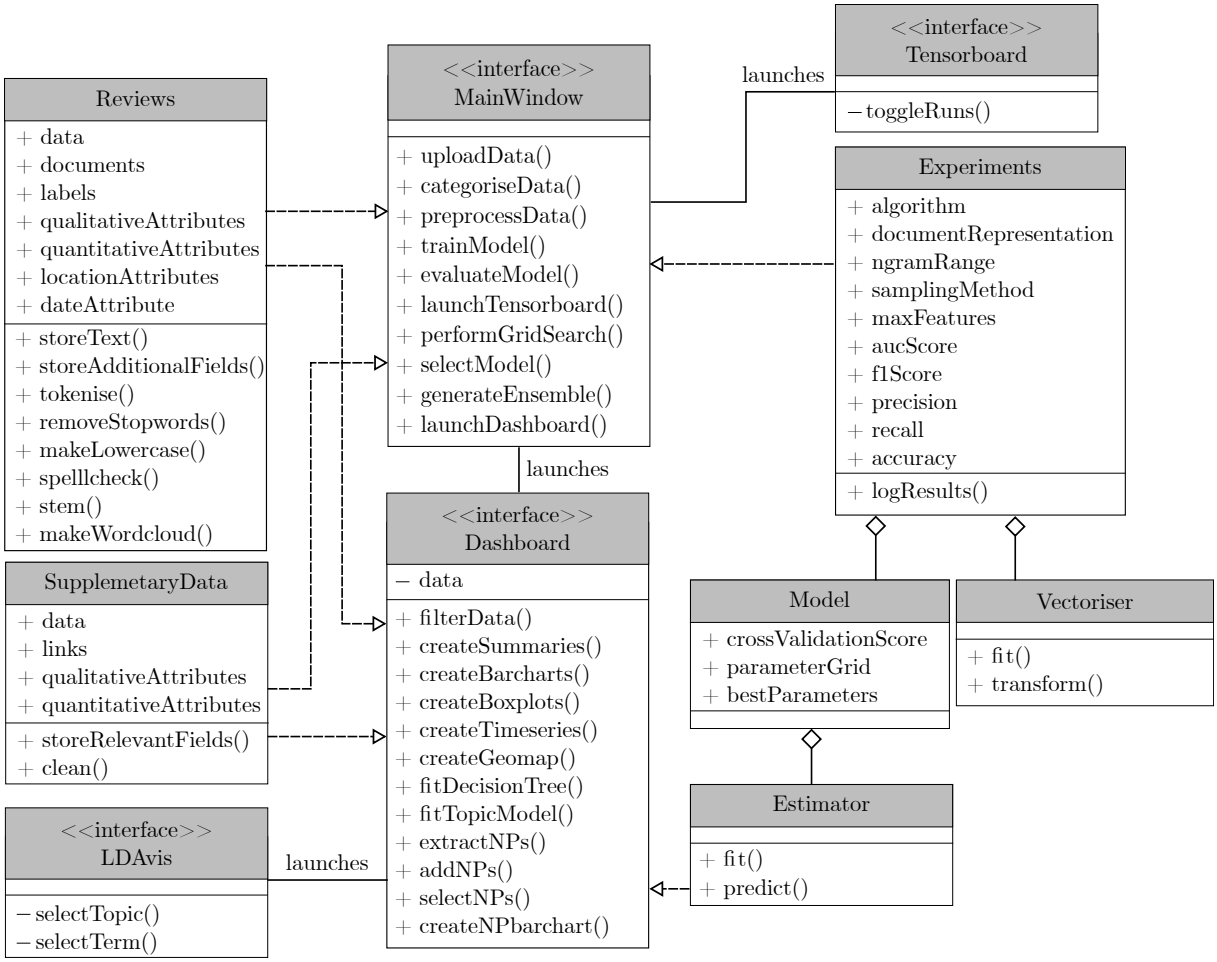


FIGURE 7.1: *The class diagram of the ECCO system.*

There are four interface classes in the ECCO system, namely the *MainWindow*, the *Dashboard*, the *Tensorboard* and the *LDAvis* interface. The former facilitates the execution of all the modules contained in the preprocessing and analysis components of the ECCO framework, such as categorising data attributes and evaluating models. In order to perform these operations, the interface makes use of the *Reviews* class and the *SupplementaryData* class, which house the data and functions related to the documents in the corpus, and the data and functions related to additional data, respectively. Furthermore, the *MainWindow* interface makes use of the *Experiments* class to evaluate models developed in the modelling component of the framework. These experiments comprise a sentiment classification model (*Model*) and a *Vectoriser* that is used to transform the text into numerical features for machine learning models. The model, furthermore, contains an *Estimator* class that may be trained on a data set and may be used to predict values for unknown data. The *Model* class is used to log the metrics related to the hyperparameter tuning of a model, whilst the *Experiments* class is used to log the performance of a model-vectoriser pair during testing.

In order to facilitate a manual hyperparameter search for deep learning methods, the *MainWindow* interface launches another interface, the *Tensorboard*, where the performance of individual network configurations is visualised. Furthermore, once the preprocessing and modelling stages have been completed, the *Dashboard* interface is launched, which facilitates the execution of the modules in the analysis component of the ECCO framework whilst making use of the *Reviews*, *SupplementaryData* and *Estimator* classes. During the analysis stage, the Dashboard interface may also launch the *LDAvis* interface, which facilitates visualisation and analysis of the LDA topic model that is fit to the data.

The class diagram in Figure 7.1 bears no information on *how* the functions in the ECCO system were implemented, but merely describes the architecture of the system. The logic underlying each of the components of the system and their interaction with one another is explained during the system demonstration in the following section. In the remainder of this section, details are relayed on the technical implementation of the ECCO system in a computer environment.

As described in §5.1.4, various design choices have to be considered when implementing a DSS. The options selected for each of these considerations are shown in Table 7.1. Since the system serves merely as a proof of concept of the ECCO framework, it was implemented offline and without any licensing restrictions. The system was developed in `Python` 3.7. `Python` was chosen since it is widely regarded as the most popular language for machine learning applications [87, 137], overtaking other languages such as R and `Java` in this field [238].

No specialised DBMS was employed, since no complicated data manipulations had to be facilitated and no large amounts of data had to be stored by the system for demonstration purposes. Data were stored, where applicable, in `DataFrame` constructs native to the `Python` library `Pandas` or arrays native to the `Python` library `Numpy`. Furthermore, in order to save models in their deployable format and to rapidly transfer files between elements in the *Dashboard* interface, `JSON` (`JavaScript` *Object Notation*) was used, which is a particularly lightweight format for storing and transporting data [332].

Design consideration	Selected option(s)
Licensing	Free
Connectivity	Offline
Language on server side	`Python`
Data store	`Pandas Dataframes`, `Numpy arrays`, `JSON` files
IDE	`Pycharm`, `QtDesigner`
Application framework	`Qt`, `Dash`, `Tensorboard`

TABLE 7.1: *Design considerations for the ECCO system.*

Three different application frameworks were employed during the development of the ECCO system. The first is `Qt` [313], which was used to develop the user interface that houses the processing component and the sentiment modelling component of the ECCO framework (referred to as *MainWindow* in Figure 7.1). The `Tensorboard` [311] framework was used to visualise the performance of deep learning models during the hyperparameter tuning process. Finally, the `Dash` [240] web application framework was used to implement the analysis component of the ECCO framework, which executed by means of a user interface (referred to as *Dashboard* in Figure 7.1) hosted on the user's web browser. These frameworks were chosen to facilitate the mentioned functions due to their relative strengths. `Qt` is well-suited for creating versatile user interfaces with complex background operations, whilst `Dash` supports interactive visualisations, but has limited memory capacity and is therefore not equipped for longer computations [39]. `Tensorboard` is a well-established framework for visualising neural networks [311] *via* interactive

graphs and was used for this purpose in order to avoid *'reinventing the wheel.'* The layout of the user interface *MainWindow* was developed using the IDE `Qt Designer` [312], whilst all other programming was conducted using `PyCharm` [138], an IDE developed specifically for the `Python` programming language.

7.2 Demonstration

In this section, the ECCO system is demonstrated. Each of the system's functions is presented from the perspective of the user and showcased using screen shots of the actual computerised system. In order to facilitate this demonstration, example data are employed pertaining to customer feedback on the services of a retail bank. A case study is performed in respect of the same data in Chapters 8 and 9.

The home page of the ECCO system is shown in Figure 7.2. It contains a list of all the functions facilitated by the system, partitioned into the familiar three subcomponents of the ECCO framework, namely processing, modelling and analysis. This is intended to provide the user with an overview of the system and an expectation of what is to follow, in line with the user interface design principle of predictability (described in §5.1.3). In the remainder of this section, the system's functions are presented in the order in which they appear in Figure 7.2, each time making reference to the respective heading in the figure.

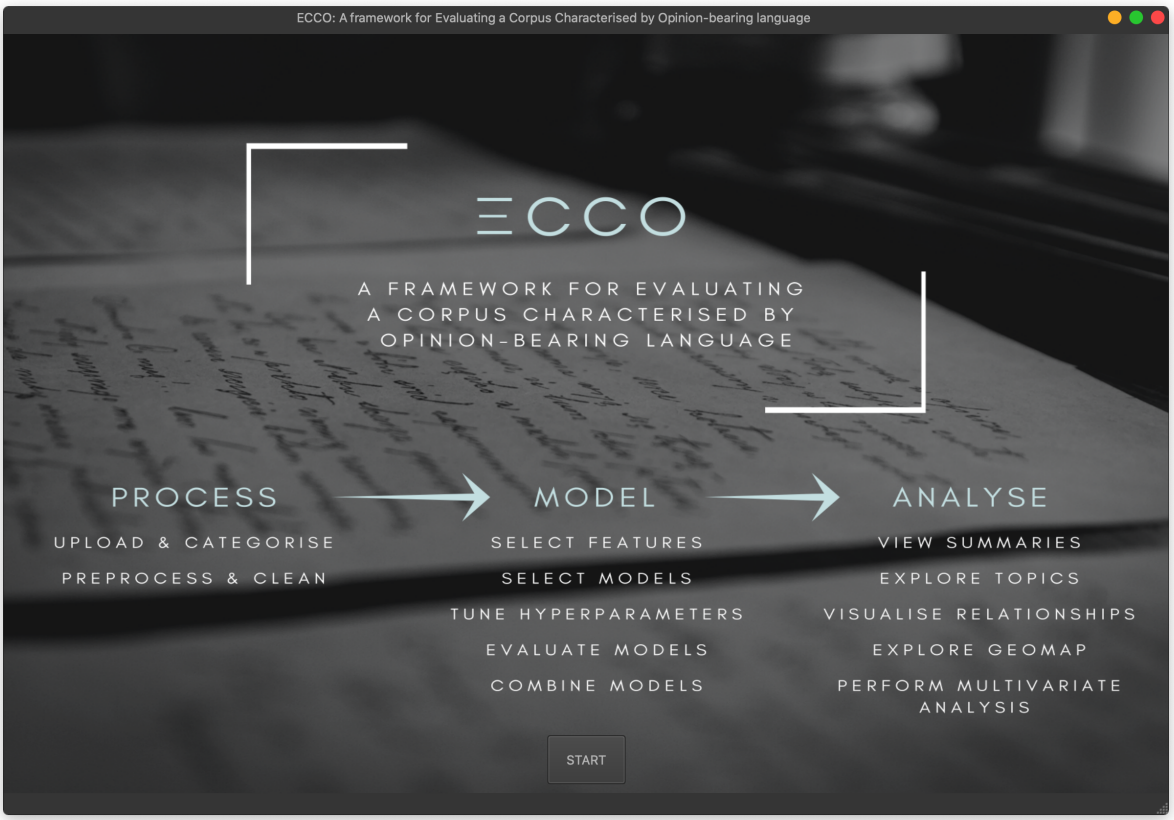


FIGURE 7.2: *The home page of the ECCO system, containing an overview of the functions facilitated by the system.*

7.2.1 Implementation of the processing component

Upon clicking the *start* button on the home page of the system, the user is redirected to the *upload & categorise* page, the first of the categories under the *process* heading in Figure 7.2. As shown in Figure 7.3, the heading of the page is consistent with the terms appearing on the home page, and the user is given an indication of his or her traversal through the system in the top left-hand corner of the page, which in the figure indicates that the user has transitioned from the *home* page to the *upload* page.

The user is now expected to upload the data that are to be analysed. As mentioned in §6.3, the data are partitioned into an unstructured component (the corpus of text documents) and a structured component (additional data stored in a relational data table) by the ECCO framework. In the ECCO system, the data to be uploaded are partitioned slightly differently. The user is requested to upload *review data* containing the corpus as well as data relating to its constituent documents and *supplementary data* providing additional information separately. In this manner, several different cases relating to the data available to the user can be accommodated. These cases are described in Table 7.2. In Case 1, the user has only unstructured text documents available (*e.g.* product reviews). In Case 2, these documents are annotated with sentiment *labels* that may be used to train a classifier (*e.g.* product reviews along with a label indicating whether the reviews are favourable or unfavourable). In cases 3 and 4, the user also has structured data available that describe the documents in more detail (*e.g.* the IDs of the customers who sent the reviews, and the dates on which the reviews were submitted). Finally, in cases 5 and 6, data are also available that describe some other document attribute (*e.g.* the name, age and gender of the customers identified by each customer ID).

	Documents	Labels	Attributes describing documents	Attributes describing document attribute
Case 1	✓			
Case 2	✓	✓		
Case 3	✓		✓	
Case 4	✓	✓	✓	
Case 5	✓		✓	✓
Case 6	✓	✓	✓	✓

TABLE 7.2: *Possible cases that may occur in respect of the data available to the user of the system.*

The user uploads two relational data sets into the system. The first is referred to as *review data* and contains the documents, labels and attributes describing the documents, where available. The second is referred to as *supplementary data* and contains attributes which describe other document characteristics. Only the review data are required in order for the system to function correctly, which is furthermore only required to contain the documents themselves — any additional data are optional.

After having uploaded the data sets by means of the file picker, which is launched when the *upload review data* or *upload supplementary data* buttons are clicked, the user can now categorise the relevant data attributes. This constitutes Module 1.0 in the ECCO framework. As was described in §6.3.1, this entails tagging *'distinct'* attributes (Module 1.1) and categorising the data types of the remaining attributes (Module 1.2).

In the ECCO system, distinct attributes related to review data include a data field describing the actual document text and, if available, a field containing the sentiment labels. In the example in Figure 7.4, these fields were selected as *Comment* and *Sentiment*, respectively, from the given

FIGURE 7.3: *The upload and categorise page of the ECCO system.*

drop down lists. These lists were automatically populated with the column headings of the data set, both for the user's convenience and as a method of input validation.

Other distinct attributes are those related to location, namely the fields containing the geographic coordinates (latitude and longitude) of the data, as well as the field containing the name of the specified location (*Branch Name* in the example in Figure 7.4). If latitude and longitude values are not available, town and province names may be specified by the user instead. These are then converted to latitude and longitude values internally by the system using an open source mapping of town and province names to geographic coordinates[1]. Finally, the field containing date values related to the documents may be selected. This attribute is expected to describe when a particular document was composed (*e.g.* when a certain product was reviewed). In the example in Figure 7.4, the attribute *CreatedDate* was designated as this distinct attribute.

In this particular instantiation of the ECCO framework, it was assumed that location and date attributes were related directly to documents, since the objective of the system is to investigate the relationship between these variables and the sentiment of the given documents. Distinct attributes related to the supplementary data include the *links* or keys which relate the review data and supplementary data tables and which may be used to merge them (as described with respect to relational databases in §5.1.1). In Figure 7.4, the fields *Contact CIF_NUMBER* in both the supplementary data set and the review data set are designated as these keys.

Finally, for both the review data and the supplementary data, qualitative and quantitative data attributes may be categorised using the *list box* widgets located at the bottom of each respective section of the upload page. These list boxes have also been populated with the column headings of the respective data sets. The user selects qualitative attributes using the list box on the left-hand side of the page and quantitative attributes using the list box on the right-hand side of the page. In Figure 7.4, for example, only qualitative attributes have been selected for the review data, including *Branch Province* and *Branch Type*, whilst the attributes *Salary* and *Age* have been categorised as quantitative attributes of the supplementary data.

When satisfied with the selection, the user can proceed to the *preprocess & clean* page, the second category under the *process* section in Figure 7.2, by clicking the *confirm selection* button. As shown in Figure 7.5, the site map indicating the user's traversal through the system in the top left-hand corner of the screen is updated, the page heading remains consistent with the terminology on the home page and the user is always permitted to go back to the previous page (*via* the button in the bottom left-hand corner of the screen) or undo an action, in line with the design guidelines discussed in §5.1.3.

The most prominent element on the page in Figure 7.5 is the *word cloud* visualising the most frequently occurring terms in the corpus. Term frequencies are indicated by the relative sizes of the words, where frequent words are shown in larger font sizes (the word colouring is arbitrary). The initial word cloud shown is intended to give the user an overview of the words in the unprocessed corpus. Upon examining the word cloud in the figure, it is evident that the most frequent words in the corpus are stop words such as *"and"* and *"it,"* which are uninformative for a classifier.

The user may then select certain preprocessing steps in the top section of the page (labelled *preprocess data*). This constitutes the *desired settings* data flow in Figure 6.5 of the ECCO framework. In particular, these settings govern which preprocessing steps in modules 2.0–4.0 of the framework (tokenisation, filtering and normalisation) are executed and, where applicable, which algorithms and parameters are used in the process. Most of the preprocessing steps are

[1] In order to facilitate this conversion process, such a mapping must be provided to the system in the form of a `.CSV` file in the program folder. By default, the system is able to accommodate South African location data.

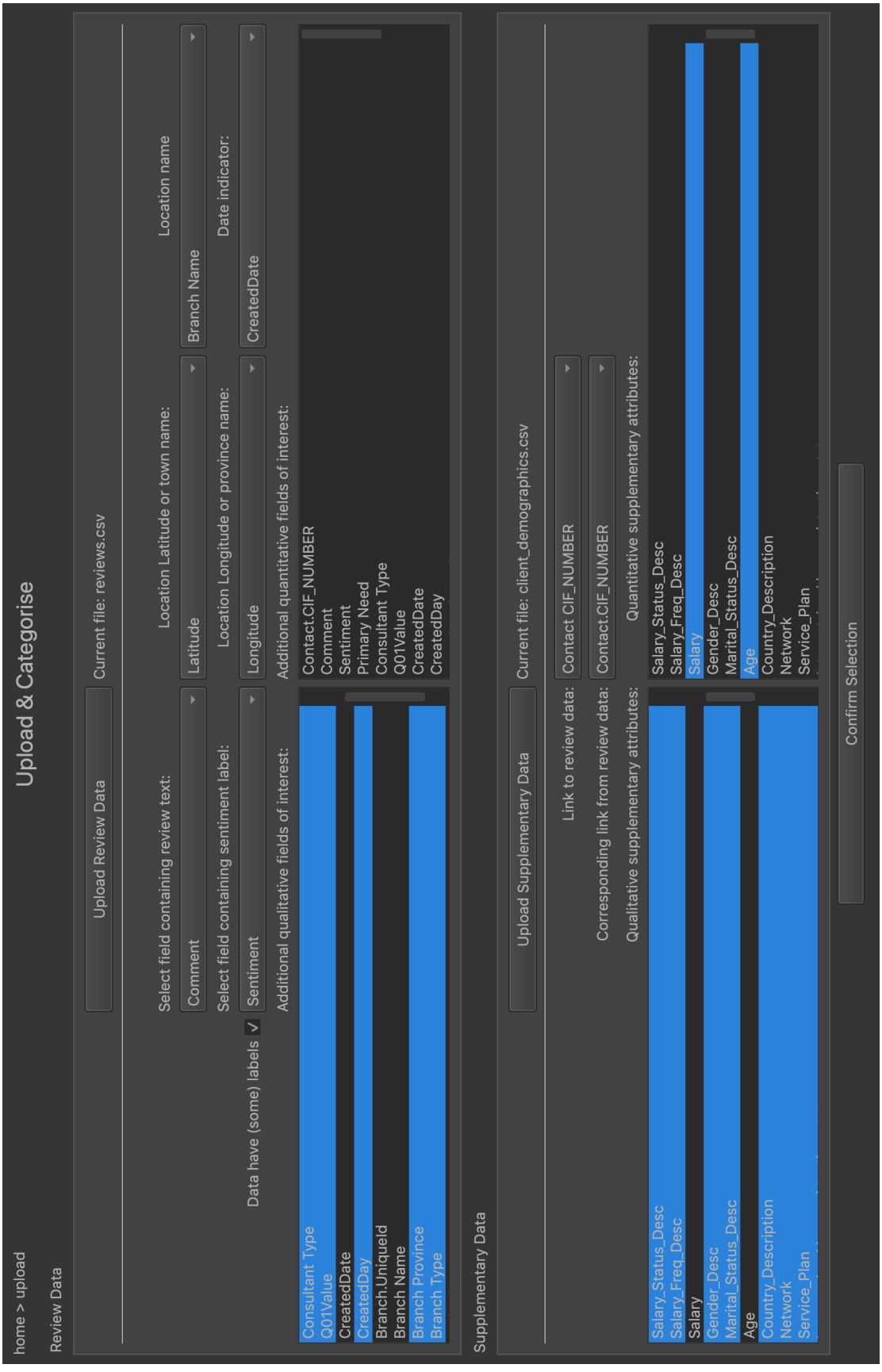


FIGURE 7.4: *The upload and categorise page of the ECCO system populated with example data.*

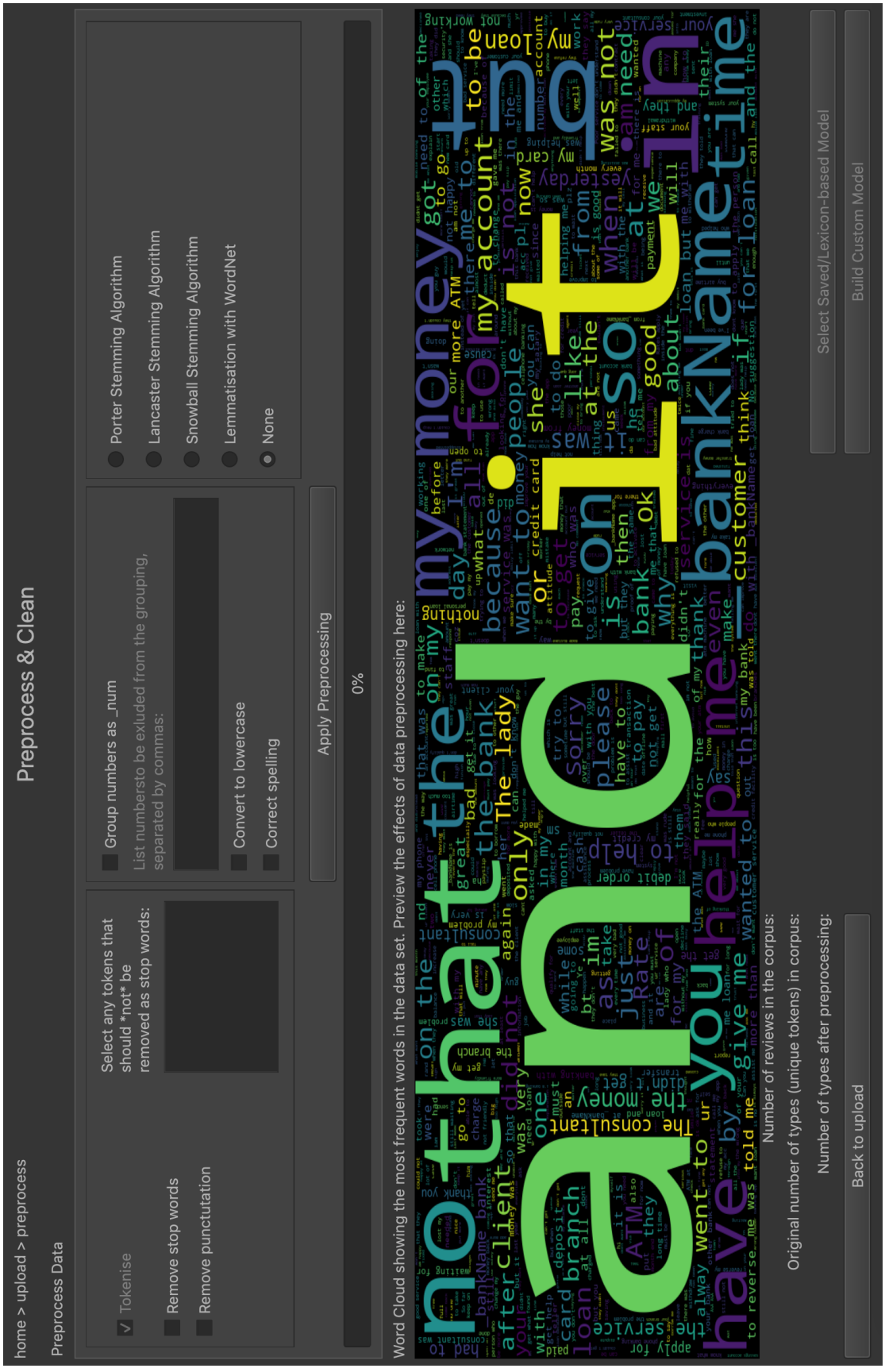


FIGURE 7.5: *The preprocess and clean page of the ECCO system.*

executed by invoking the relevant functions in the well-known NLTK library [29] for natural language processing.

As shown in Figure 7.5, the tokenisation procedure (Module 2.0) is not optional, since this step is required for most of the other preprocessing steps and most of the models employed later in the system. The user is, however, able to choose whether to remove stop words and punctuation (Module 3.1). If either of these options is selected, the list box in the left-most block is populated with a standard stop word and/or punctuation list from the NLTK library. This list is, however, not customised for sentiment analysis tasks. The user may, therefore, choose to remove certain elements from the list by selecting them in the list box. As discussed in §6.3.1, such elements may include negation words and words or punctuation marks that are indicative of sentiment.

Similarly, the user may choose whether to group tokens containing numbers into a single category designated as *_num* (Module 3.2). As before, the user may also opt to exclude certain numbers from this process that bear some meaning in the specific context, this time by entering these numbers in the input field provided.

Concerning the normalisation of the text, the user is able to selectively convert the text to lowercase (Module 4.1), correct the spelling of individual words (Module 4.2) and perform stemming or lemmatisation (Module 4.3). In the latter case, the user can select either lemmatisation or one of the available algorithms for stemming. Three popular stemming algorithms, namely Porter's Stemming Algorithm, the Snowball Stemming Algorithm and the Lancaster Stemming Algorithm were selected to populate the framework in this case. In the natural language processing community, Porter's Stemming Algorithm (described in §4.1.1) is generally regarded as the most *gentle* algorithm, leading to more intuitive results at the expense of a larger computational cost. The Snowball Stemming Algorithm (also known as *Porter2*) is regarded as an improvement over the first, with faster computational time and a slightly more *aggressive* approach. Finally, the Lancaster Stemming Algorithm is viewed as the most *aggressive* of the three, which typically reduces the number of unique tokens in a corpus significantly and in a short period of time [295]. Lemmatisation (described in §4.1.1) is performed in this system using the lexical database WordNet, which was introduced in §4.2.3.

When the user has made the desired selections, he or she clicks the *apply preprocessing* button and the selected preprocessing steps are executed. The text is first tokenised, and each of the tokens is then converted to lowercase. Subsequently, stop words and punctuation are removed, taking into consideration the tokens excluded from the stop word lists. The spelling correction and grouping of uninformative numbers is then performed according to a custom algorithm. Finally, each token is subjected to the selected stemming or lemmatisation algorithm.

The custom algorithm, which executes both spelling correction (Module 4.2) and grouping of uninformative numbers (Module 3.2) simultaneously is given in pseudo-code form in Algorithm 7.1. The popular *Aspell* [12] spell checking engine is used in this process, and more specifically its functions check(w) and suggest(w). These functions return a boolean value indicating whether the word w is contained within Aspell's internal dictionary and a list of suggested corrections, respectively. Furthermore, the frequency distribution class from the NLTK library is employed, which counts the frequency of all tokens occurring in the corpus. This distribution is stored in the system as $corpus.freq$. For any word w, its normalised frequency in the corpus can then be extracted using $corpus.dist.freq(w)$.

The algorithm first checks whether spelling correction has been selected by the user. If so, it then iterates through each token in each document in the corpus. Each token is first examined for the number of numerical characters it contains. If it contains more numerical than non-numerical characters, it is treated as a number. This distinction allows currency values such as

Algorithm 7.1: Process used for spelling correction in the ECCO system.

Input : A tokenised corpus with misspellings, boolean values X and Y, indicating whether spell checking and number grouping are selected, respectively, and a list L detailing numbers not to be grouped by the algorithm.

Output: A tokenised corpus with corrected misspellings and substituted numbers.

```
1  if X = TRUE then
2      for each document in the corpus do
3          for each token t in the document do
4              if most characters in t are numerical then
5                  if Y = TRUE then
6                      if L is empty or if t not in L then
7                          t ← '_num';
8              else
9                  if aspell.check(t)= FALSE then
10                     s ← aspell.suggest(t);
11                     if lowercase(t) = lowercase(s_i) for any i ∈ {1,...,len(s)} then
12                         t ← s_i
13                     else if corpus.dist.freq(t) < corpus.dist.freq(s) then
14                         t ← argmax(corpus.dist.freq(s));
15 else if Y = TRUE then
16     for each document in the corpus do
17         for each token t in the document do
18             if most characters in t are numerical then
19                 if L is empty or if t not in L then
20                     t ← '_num';
21 return Corpus
```

"R100" to be treated as numbers whilst shorthand notations and typos such as "*gr8*" or "*2yes*" are treated as text. If numerical grouping was also selected by the user, numbers that are not contained in the list of numbers to be excluded by the user are replaced by the generic token *_num*.

Tokens that are treated as text are then checked for spelling errors by the Aspell engine. If incorrect spelling is detected, a list of suggestions to correct the spelling error are extracted using Aspell. Subsequently, a decision must be made as to which suggestion, if any, should be accepted to replace the current token. In order to determine this, a simple check is first performed to determine whether the misspelt word is the same as one of the suggestions when case is disregarded (Line 11 of the algorithm). If the word "*amazing*", for example, were accidentally typed as "*amAzing*," the suggestion "*amazing*" would immediately be accepted. If this is not the case, the frequencies of the current, apparently misspelt token and each of the suggestions in the corpus are retrieved. The token that is most frequently used in the corpus is then selected as the correct spelling[2]. This allows the spell checking algorithm to correct the spelling of a word with consideration of the context in which it was used. If, for example, names or jargon are frequently used in the corpus that are flagged as incorrect by Aspell (*e.g. "Stellenbosch"* or "*hyperparameters*"), such words are not corrected. Finally, lines 15–20 in Algorithm 7.1 execute the same process for grouping numerical features as before, in the case where spelling correction is not also selected by the user.

After modules 2.0–4.0 have been executed according to the user-specified settings, the word cloud is updated to show the processed state of the data. Furthermore, a summary is shown of the number of documents (or *reviews*) in the corpus, the number of unique tokens (*types*) in the data before preprocessing was applied, as well as the number of types present after preprocessing the data. This constitutes the *effect summary* data flow returned to the user in Figure 6.5. An example of this is shown in Figure 7.6.

In this case, the user selected all possible preprocessing steps and decided to exclude terms such as "*no*", "*nor*" and "*not*" from the stop word list, and the numbers 1, 2 and 3 from the numerical grouping process. Furthermore, the *lemmatisation with WordNet* option was selected. The summary in the figure shows that the number of unique tokens in the corpus was reduced by more than half as a result of the preprocessing steps. The most frequent terms in the corpus have also changed. Terms expressing numerical quantities and negation words are understandably large since the former is a grouped feature and the latter also constitutes stop words. Looking more closely at the other large words, however, gives the user some more insight. The terms *"loan,"* *"bank,"* *"money,"* *"consultant"* and *"problem,"* for example, are also prominent. Furthermore, the words in the word cloud are all complete words that can be found in the dictionary. If a stemming algorithm was chosen over the lemmatisation algorithm, the expected output would contain many truncated words.

The preprocessing activities described up to this point relate to the documents uploaded in the *review data* section by the user. Preprocessing activities that relate to the structured, supplementary data, namely Module 5.0 of the ECCO framework, are executed as background activities. As explained in §6.3.1, this is the case since the focus of the framework is the analysis of unstructured data and the structured data are assumed to be extracted from a well-maintained database. Nevertheless, duplicate removal (Module 5.1) and the treatment of missing values (Module 5.2) are performed on the supplementary data. More specifically, any rows in the data set that occur more than once are first removed. Thereafter, rows or columns for which more than

[2]As indicated by the strict inequality in Line 13 of Algorithm 7.1, the token appearing in the original review is retained in the case of a tie.

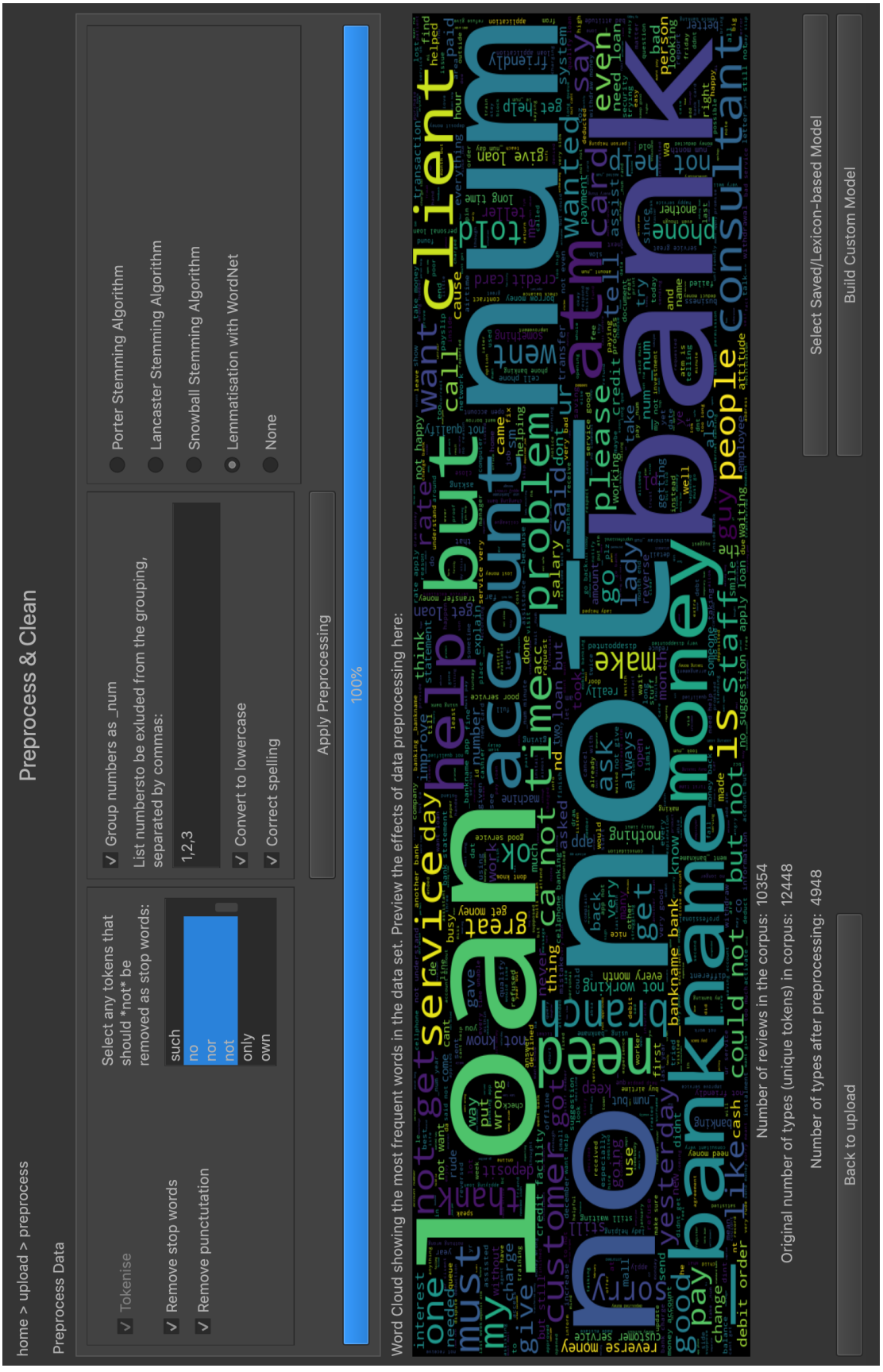


FIGURE 7.6: *The preprocess and clean page of the ECCO system populated with example data.*

a certain percentage[3] of the data are missing are also removed. Any missing entries that remain after these operations are *imputed* with the average column value (for quantitative variables) or the most commonly occurring value in the column (for qualitative variables). An effect summary is then returned to the user in the form of a pop-up window as shown in Figure 7.7. From this summary the user can deduce that there were 15 364 (277 713 − 262 349) duplicate entries in the data set, and that 5 740 (262 349 − 256 609) records were missing more than 50% of their entries.

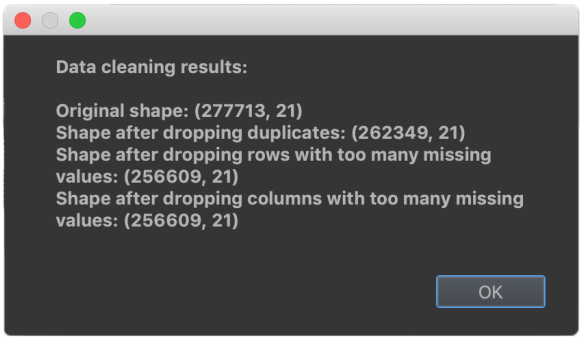


FIGURE 7.7: *Pop-up window informing the user of the effects of the data cleaning process.*

7.2.2 Implementation of the modelling component

Once the user is satisfied with the outcome of the preprocessing phase, he or she can proceed to the sentiment modelling phase. As is shown at the bottom right-hand corner of Figure 7.6, there are two possible options for approaching this phase. The first option entails selecting either an *off-the-shelf* lexicon-based model with a built-in sentiment lexicon, or a model that has previously been developed using the ECCO system and saved on the user's computer. This option is useful in cases 1, 3 and 5 of Table 7.2 where sentiment labels are not available for any of the data. Furthermore, this also enables model development and model analysis to be executed separately. If sentiment labels were to be provided for some (or all) of the data, the user may also choose the second option: To develop a *custom model* tailored to the data at hand, as per the process outlined in the modelling component of the ECCO framework. In either case, the aim is to perform a document-level sentiment polarity classification, where documents are classified as *positive*, *negative* or, optionally, *neutral* (see §4.2.2 for a detailed definition).

If the first option is selected, the user is redirected to the *select existing model* page shown in Figure 7.8. The user is then given a choice between four different lexicon-based models that have been embedded into the ECCO system, namely *Sentiwordnet*, *Pattern*, *Vader* and the *Hu and Liu Opinion Lexicon*. Each of these models is already fitted with a sentiment lexicon and can therefore readily be applied to input data.

Sentiwordnet [82] is a lexical resource in which each synset in the WordNet database (see §4.2.3) is assigned an *objectivity score*, a *positivity score* and a *negativity score*. In the ECCO system, the polarity score of a token is computed by subtracting the negativity score from the positivity score of the synset to which the token most likely belongs[4]. The scores for all tokens in a

[3] In this particular implementation of the ECCO system, 50% was arbitrarily chosen as the threshold in an attempt to equally balance the conflicting objectives of reducing the loss of data [76] and accurately representing the data [272].

[4] A word can belong to several WordNet synsets, which are distinguished by their contextual meaning. By selecting the first synset suggested by the database, complicated word sense disambiguation procedures were avoided.

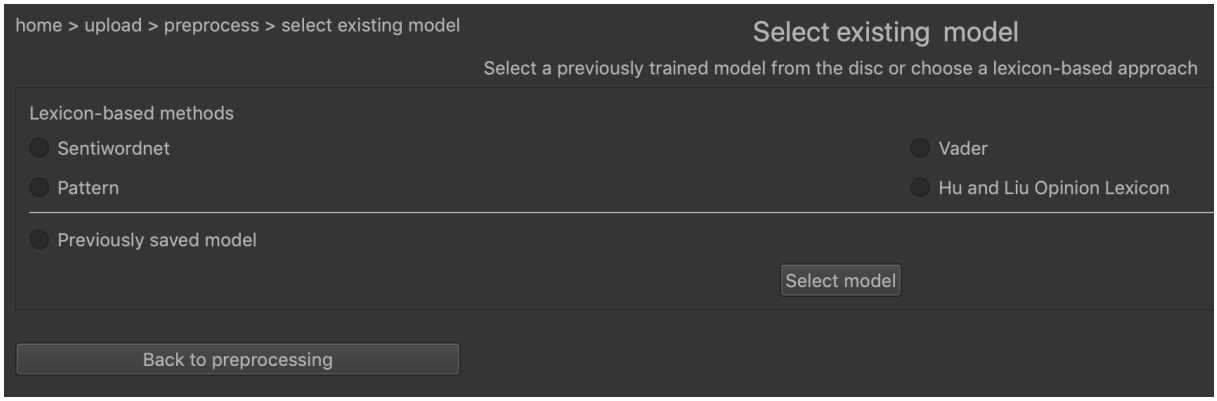


FIGURE 7.8: *The select existing model page of the ECCO system. (The image was cropped to show only the important elements on the screen.)*

document are then summed to yield a total score for the document. As per the recommendation in the documentation of the `Pattern` library [58], which is used to implement the Sentiwordnet model, the document is classified as *negative* if the total score is negative, as *positive* if the total score is greater than 0.1 and as *neutral* otherwise. The same rule is applied to determine a document's sentiment class based on its polarity score according to the *Pattern* model, which refers to the built-in sentiment model of the `Pattern` library. This model takes as input the entire document; therefore, an aggregation of individual token scores is not necessary.

Vader [131] (*Valence Aware Dictionary for sEntiment Reasoning*) is a rule-based model which assigns a sentiment score to documents based on a lexicon of 7 500 empirically evaluated features commonly occurring in social media blogs, as well as generalisable heuristics used by humans. As proposed in the original paper [131], these sentiment scores are transformed into sentiment categories using the threshold values of -0.05 and $+0.05$, where documents with sentiment scores below the negative threshold are classified as *negative*, those with sentiment scores above the positive threshold are classified as *positive*, and the remainder are classified as *neutral*.

Finally, the opinion lexicon developed by Hu and Liu [129] contains a list of adjectives along with their sentiment polarities (*positive*, *negative* or *neutral*). This lexicon is employed in the ECCO system in conjunction with a simple counting rule: Documents containing more positive than negative adjectives in the lexicon are classified as *positive*, documents with an equal number of positive and negative adjectives are classified as *neutral*, and the remaining documents are classified as *negative*.

In the case of binary classification, documents that would be classified as *neutral* according to the rules outlined above are assigned the *negative* class label. Moreover, for the purposes of model evaluation, lexicon-based models are treated as discrete classifiers. Although their classification is based on the magnitude of numerical sentiment scores, these scores are not bound to a fixed range and therefore cannot be interpreted as probabilities.

Instead of selecting one of these four lexicon-based models, the user may also choose to retrieve a *previously saved model*. In this case, the user is prompted to locate the model file on the computer *via* a file picker interface that is launched when the *select model* button is clicked. The selected model, whether it is a lexicon-based model or a model loaded from the disc, is then deployed to classify the sentiment polarity of the documents that were uploaded by the user. Subsequently, the *Dashboard* interface is launched to facilitate an analysis of the model's results, as described in the next section.

If the option to develop a custom model is selected, on the other hand, the user is redirected to the *develop models* page of the ECCO system. As shown in Figure 7.9, this page is organised into a large section in the middle of the screen and a smaller section at the bottom of the screen labelled *training*. The larger section is partitioned into four *tabs*, the first of which facilitates feature engineering in accordance with the first heading under the *model* category in Figure 7.2. The remaining tabs relate to the *select models* heading under the same category, and facilitate the selection of '*shallow*' machine learning models, deep learning models and lexicon-based models, respectively. In each tab, the user is afforded the ability to select various features or models for comparison. Furthermore, the user is able to initiate a grid search for the purpose of hyperparameter tuning by specifying several values for a model's hyperparameters. The configuration of this grid search is controlled *via* the *training* section at the bottom of the page.

In the remainder of this section, the contents of these tabs are described in respect of the feature configurations and models chosen to populate the ECCO framework for the proof of concept demonstration, as well as the associated hyperparameters that may be tuned using the ECCO system. Selecting and configuring these features and models constitutes processes 6.0, 7.0 and 9.0 of the ECCO framework. Subsequently, the implementation of these models (Process 8.1 and 10.0) and the hyperparameter tuning process (Process 8.2) are discussed with reference to possible configurations that may be set within the *training* section of the page in Figure 7.9. Finally, a comparative evaluation of the models generated using the ECCO system is carried out (Process 11.0). During this phase, several models may also be combined to form an ensemble with a view to improve generalisability (Process 12.0).

The active tab in Figure 7.9 is the *feature engineering* tab, which is partitioned into two sections responsible for feature generation (Module 7.1 of the ECCO framework) and feature selection (Module 7.2 of the ECCO framework), respectively. In this particular instantiation of the ECCO framework, variations of the bag-of-words model (see §4.1.1 and §4.1.2) were implemented for the purpose of feature generation. Various *n-gram ranges* may be specified, which control the length n of the word sequences that are extracted from each document as features. More specifically, unigrams, bigrams or trigrams may be selected, as well as several combinations of these features (*i.e.* unigrams and bigrams, bigrams and trigrams and unigrams, bigrams and trigrams). Furthermore, up to three different *document models* may be selected, which determine how the extracted word sequences are transformed into numerical vectors (see §4.1.2). These vectors may represent whether or not an n-gram appears in the given document (*term presence*), how often it appears (*term frequency*) or how much information the n-gram is likely to carry based on its usage across all documents (according to its *TF-IDF* normalised frequency count). Finally, one feature selection method is applied in order to reduce the size of the resulting vocabulary. This method is a variation of frequency pruning (see §4.1.2), where only the x most frequently occurring words in the vocabulary are retained[5] (here x is a user-specified parameter).

Upon clicking the *evaluate and compare* button at the bottom right-hand corner of Figure 7.9, each of the selected n-gram ranges is combined with each selected document model. Each of these combinations is then implemented along with the feature selection step as an object of the *Vectoriser* class shown in Figure 7.1, which was implemented using the `CountVectorizer` class of the `Scikit-learn` library [236]. The selection shown in Figure 7.9, for example, would produce nine (3×3) instances of the *Vectoriser* class, the first of which (the combination of the unigram term presence feature generation model and the feature selection model with $x = 250$) transforms each document into a binary vector of maximum length 250, where each entry in the vector represents the presence or absence of a certain unigram in the vocabulary.

[5]Note that the most frequent terms in the original corpus, the stop words, have likely already been removed from the vocabulary during the preprocessing phase.

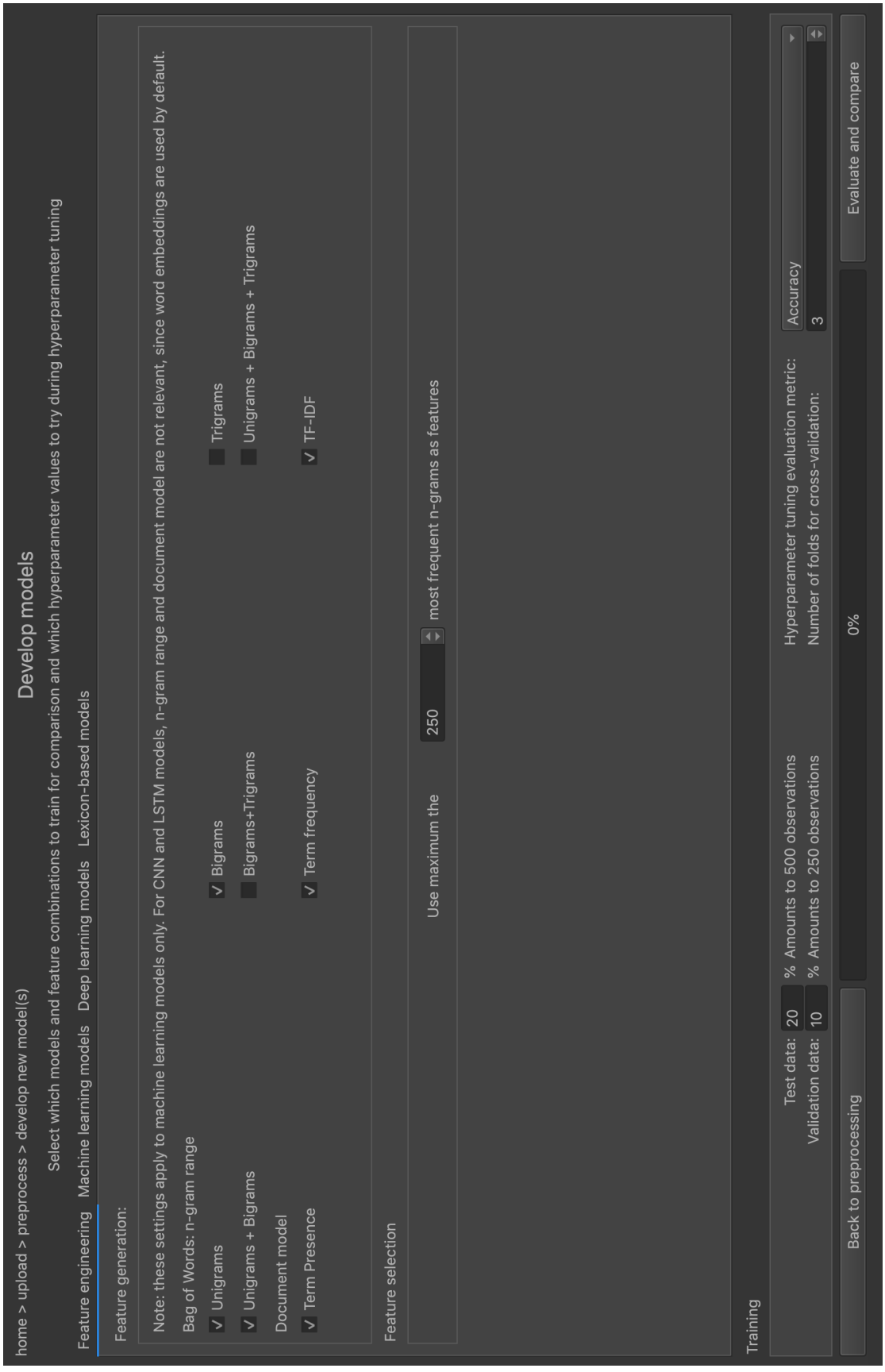


FIGURE 7.9: *The develop models page of the ECCO system with the feature engineering tab active.*

As is explained to the user in Figure 7.9, the *Vectoriser* objects are employed to transform the documents to feature vectors used as input for machine learning models[6]. In the following two tabs, these models are configured. The three *traditional* algorithms identified in 4.2.3 were implemented to represent '*shallow*' machine learning models in the ECCO system, namely naïve Bayes, SVM and multi-class logistic regression (maximum entropy). Each of these algorithms was implemented using the appropriate *Estimator* class in the `Scikit-learn` library. As shown in Figure 7.10, the user may choose whether to include each of these algorithms in the comparative evaluation by checking the check box next to the algorithm name. Furthermore, the user may specify values for the algorithms' hyperparameters.

For the simple naïve Bayes algorithm (see §3.3.4), the only hyperparameter is the smoothing parameter α, which is employed by the Bernoulli naïve Bayes classifier and the multinomial naïve Bayes classifier. The Gaussian naïve Bayes classifier, on the other hand, does not make use of any hyperparameters. The type of classifier is selected in accordance with the document representation model selected in the feature engineering tab. More specifically, the term presence, term frequency and TF-IDF document representations are used in conjunction with the Bernoulli, multinomial and Gaussian naïve Bayes classifiers, respectively.

For the SVM algorithm (see §3.3.3), the penalty parameter C may be tuned to control the bias-variance trade-off of the model. Furthermore, the kernel type may be selected as a linear kernel, radial kernel, sigmoid kernel or polynomial kernel with degree one (*P1*), degree two (*P2*), or degree three (*P3*). The kernel coefficient γ, which is used in combination with the radial kernel, is also treated as a hyperparameter.

Finally, the hyperparameters that may be tuned for the logistic regression algorithm[7] (see §3.3.5 and §3.3.6) relate to the minimisation of the cost function, which includes an ℓ_2 regularisation (see §3.5.1) term by default. More specifically, values for the *inverse regularisation strength* C may be specified, which is the coefficient for the ℓ_2 regularisation term. Furthermore, the optimisation algorithm employed and the maximum number of iterations performed by this algorithm also constitute hyperparameters. In this case, a variant of gradient descent (see Algorithm 2.2), the *stochastic average gradient* (SAG) algorithm, may be employed, as well as two second-order gradient-based optimisation algorithms, namely the Newton-CG method and the LBFGS algorithm (see §2.4.1).

Each of the tabs in the ECCO system is populated with an arbitrary (but valid) default selection of hyperparameter values that may be altered by the user. This is intended to simplify the data input process for the user and to offer guidance in terms of the required input format and the valid range of values. In the example in Figure 7.10, for example, the user has entered six values for the smoothing parameter α ranging from 0.0001 to 1. Similarly, a range of values has been entered for the penalty parameter C and the kernel coefficient γ of the SVM algorithm, as well as for the inverse regularisation strength C of the logistic regression algorithm. Furthermore, four different kernel functions for the SVM algorithm and three different optimisation algorithms for the logistic regression algorithm have been selected. The resulting number of possible hyperparameter combinations is therefore six for the naïve Bayes algorithm, thirty six ($3 \times 3 \times 4$) for the SVM algorithm and twelve (4×3) for the logistic regression algorithm. Each of these combinations is tested during the grid search and the hyperparameters of the best performing model are then employed by the *final* model for each respective algorithm.

[6] As is described in the figure, the feature generation step is not relevant for the CNN and LSTM models which instead make use of word embeddings. The vocabulary size entered in the feature selection section is, however, applied for all machine learning models.

[7] Multi-class logistic regression (maximum entropy) is henceforth referred to simply as *logistic regression*.

FIGURE 7.10: *The machine learning models tab of the develop models page.*

Three types of deep learning algorithms were also chosen to populate the ECCO framework, namely a feedforward neural network (referred to simply as *ANN* in the ECCO system), a CNN and the recurrent LSTM network (see §3.5.3 for the descriptions of these architectures). As is shown in Figure 7.11, each of these models has a considerable amount of hyperparameters that may be tuned using the ECCO system. Based on the values of these hyperparameters, the selected networks are generated using the high-level deep learning library `Keras` [151], which runs '*on top of*' the widely used `Tensorflow` [310] library. Where possible, the configurable hyperparameters are visually partitioned into three groups.

The first group contains hyperparameters related to the structure of the network. For the ANN, this includes the number of hidden layers and the number of neurons in each hidden layer, as well as the activation function applied to the output of each layer. The user may, as before, enter several configurations for each hyperparameter to be tested during the grid search. For example, if "5,5;100" were to be entered into the *neurons per hidden layer* field, the system would recognise two different network architectures to be tested[8]. The first comprises two hidden layers with five neurons each, and the second comprises one hidden layer with a hundred neurons. Three different activation functions may be chosen, namely the ReLU function, the sigmoid function and the *tanh* function (see §3.5.2). In the example in Figure 7.11, the user has selected a single network architecture comprising two hidden layers with ten neurons each and the ReLU activation function. The resulting network takes as input the feature vectors generated by the associated *Vectoriser* class configured in the *feature engineering* tab of the system (in this case a vector of length 250 for each of the n input documents). This $n \times 250$ input matrix is multiplied by a 250×10 weight matrix to produce an $n \times 10$ matrix. After a bias term has been added and the activation function has been applied, the process is repeated for the second hidden layer. Finally, the output layer (see §3.5.2) is applied, which transforms its input from the hidden layer to an $n \times 3$ matrix, adds a bias term and applies a softmax function to yield a normalised score for each of the three sentiment classes (*positive*, *negative* and *neutral*) for each document. The final predicted sentiment class is chosen as the class with the largest score.

As is described at the top of the central section of the page in Figure 7.11, the CNN architecture has the following structure. The input is first passed to an *embedding* layer (see §4.1.2), which encodes each of the tokens in the vocabulary to a numerical vector of a user-specified length *via* a mapping (or *lookup table*) learnt during training. Each document is then represented as a sequence of embedding vectors representing the tokens observed in the document in the order in which they appear. Tokens not observed during training time are represented by zero vectors. Subsequently, a number of *convolution-activation-pooling* layer groups are applied to this embedded representation according to the configuration selected by the user. In respect of the convolutional layer, the kernel size (which relates to the n-grams extracted from the data), the stride (which governs whether any n-grams are skipped during the convolution) and the number of filters applied in each layer are specified. Furthermore, the *convolution type* is selected as either a valid convolution or a same convolution, and the activation functions are selected from the same set employed for the ANN architecture. The pooling layer, which may be applied optionally, is furthermore specified in terms of its filter size and stride. If 2×2 pooling is applied, for instance, the output of each convolutional layer is reduced to half its size before being passed on to the next layer. Finally, the resulting output is *flattened* (converted to a one-dimensional representation) and a fully connected output layer is applied similar to the one used in the ANN architecture.

[8]The format in which these configurations are to be entered into the system is illustrated by means of the default input and is explained by means of a *tool-tip text* whenever the user hovers above an input field.

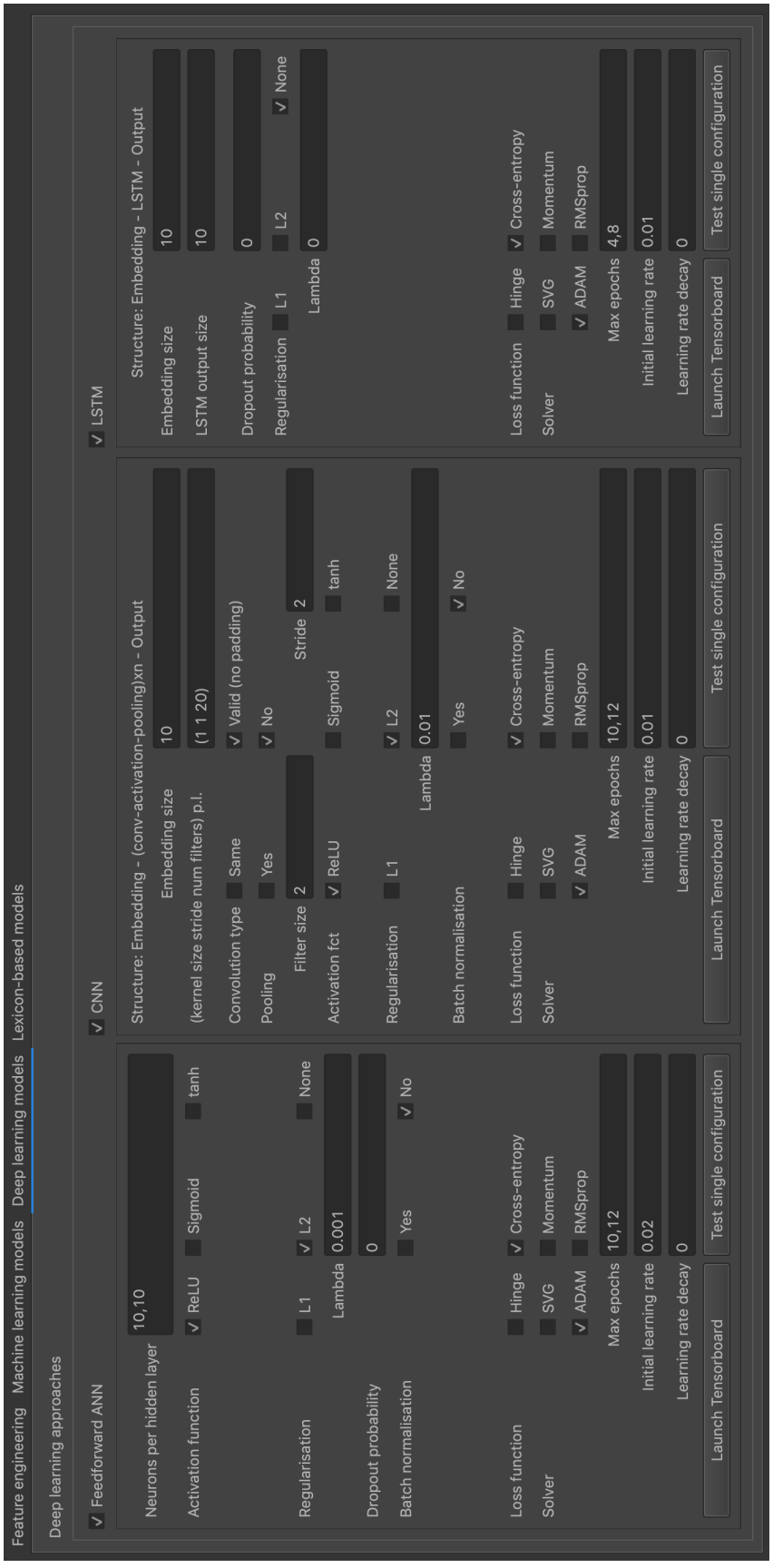


FIGURE 7.11: *The deep learning models tab of the develop models page.*

The configuration chosen by the user in Figure 7.11, for example, would result in each word in the vocabulary being embedded into a vector representation of length 10. Since the vocabulary size was limited to 250 in the *feature engineering* tab, the word embedding matrix has the dimensions 250×10. For each document, twenty feature maps are then produced during the convolution, where each token's word embedding vector is multiplied by the kernel, resulting in a feature map of size[9] $1 \times L$ where L is the maximum document (or sequence) length[10]. Since pooling is not selected by the user, the output of the convolutional layer is flattened and passed to the output layer immediately after the ReLU function is applied.

Finally, the hyperparameters governing the structure of the LSTM model are the *embedding size*, defined as in the CNN model, and the *LSTM output size*, which determines how many output units h_t are produced by the LSTM cell (see Figure 3.29). For the input entered in Figure 7.11, an embedding layer produces embedding vectors of length 10 as before, and for each sequence of embedding vectors (for each document) passed through the LSTM cell, ten output values are returned. A fully connected output layer is then applied to the output of the LSTM cell as defined previously.

The second group of hyperparameters governs the regularisation and stability of the neural networks. In particular, two different regularisation techniques may be applied, namely ℓ_1 or ℓ_2 parameter norm penalties and dropout (see §3.5.1). In these cases, the coefficient of the regularisation term λ and the dropout probability may be specified, respectively. Furthermore, batch normalisation (see §3.5.2) may be applied to reduce the network's sensitivity to weight initialisation. By default, Xavier initialisation (see §3.5.2) is employed for all layers apart from the embedding layers, where initial weights are drawn from a uniform distribution within the range $[-\sqrt{3}/n, \sqrt{3}/n]$, where n is the number of input observations. This set of hyperparameters is replicated for each of the three network types in the ECCO system, with two exceptions. Dropout is not given as an option for CNNs as it is typically not applied to convolutional layers in practice [92, 257, 315]. Since weight sharing is applied in convolutional layers, standard dropout techniques do not have the same effect with respect to convolutional layers as they do in respect of fully connected layers [257]. In fact, studies have found the standard application of dropout to these layers to be ineffective [92, 315]. Similarly, batch normalisation is not offered as an option in respect of LSTMs in view of findings that this method is challenging to implement for RNNs and can potentially be harmful to performance [167] unless a reparameterisation of the LSTM model is performed [59]. Where batch normalisation is applied, the moving mean and moving variance of the batches during training are updated at every training iteration t as per the formula $\hat{x}_t = (1 - \beta)x_t + \beta\hat{x}_{t-1}$, where the default momentum value $\beta = 0.99$ is used.

Finally, the third set of hyperparameters relate to the training of the networks (see §3.5.1) and include the loss function employed for training and the optimisation algorithm by which this loss function is minimised, as well as the number of training epochs completed by the optimisation algorithm, its initial learning rate and its learning rate decay parameter. More specifically, the user may choose to optimise either the hinge loss or cross-entropy loss using SVG, SVG with momentum, ADAM or RMSprop as the optimisation algorithm. A range of values may then be tested for the number of epochs and the initial learning rate, as well as the *learning rate decay*, where the latter is defined as the parameter λ in the function $\epsilon_{t+1} = \epsilon_t/(1 + \lambda t)$ by which the learning rate ϵ for the $(t + 1)^{th}$ training iteration is determined. The remaining hyperparmeters were taken as the default values in the `Keras` library. More specifically, a batch size of 32 is employed for all optimisation algorithms, whilst a momentum coefficient $\beta = 0.9$ is employed

[9] Since a kernel size of one was chosen, the selection of valid or same convolutions has no effect. If a larger kernel size were to be selected, however, the output size would be smaller than L for a valid convolution.

[10] Here L was chosen as the maximum document length observed during training. Documents of length $\ell < L$ are padded with zero vectors whilst documents of length $\ell > L$ observed during testing are truncated.

for the SGD with momentum, RMS Prop and ADAM optimisation algorithms. Furthermore, a variance term $\beta_2 = 0.999$ is used for the latter algorithm.

Due to the sheer number of hyperparameters that may be set for the deep learning algorithms, and the vast range of possible values that some of these parameters can assume (for instance the number of hidden layers or the learning rate), it would be computationally expensive to test many combinations of hyperparameters using the grid search facilitated by the ECCO system. Instead, it is recommended that the user first perform a manual hyperparameter search in order to obtain a rough estimate of *good* values for the hyperparameters of each network type in respect of the data at hand, so as to reduce the number of combinations evaluated during the grid search. In order to facilitate this process, a *Tensorboard* interface may be launched for each of the three deep learning algorithms using the *launch Tensorboard* button at the bottom of each respective network's section on the page in Figure 7.11. The user can then click the *test single configuration* button to evaluate the training and validation performance of the model trained using a single set of hyperparameters[11]. An example of the resulting output for an ANN model is shown in Figure 7.12.

As can be seen on the right-hand side of the figure, the model accuracy and the value of the loss function are plotted as a function of the training epochs for both the training data and the validation data. The proportion of the data used for validation is specified by the user in the *training* section of the *develop models* page, as shown in Figure 7.9. In the example in the figure, 10% of the data are used for this purpose, amounting to 250 observations. On the left-hand side of the *Tensorboard* interface, the user can change the settings for the graphs, including the degree to which the graph is *smoothed* and which *runs* are displayed. The smoothing algorithm applied is a simple moving average in which each point p in the graph is replaced by the arithmetic mean of the values in the range $[p - \lfloor w/2, p + \lfloor w/2]$, where w is the window size determined by a value entered by the user *via* the *slider* widget. If a value of 0.6 is selected, for example, 60% of the data are used as the window. The original, unsmoothed lines are *'ghosted'* in the background in a more transparent line of the same colour. The training runs may be selected by means of the check boxes and filtered by means of a *regular expression* ('*regexp*') describing the name of the run. Each run is named after the year, month, day, hour, minute and second it was created, followed by the designation of *training* or *validation*. The expression '*validation*' may, for example, be entered into the relevant field to filter out the training performance of the model.

In the example in Figure 7.12, it may be deduced from the graphs that the network is, in fact, *learning* during training, since the training loss is decreasing and the training accuracy is increasing as a function of the number of training epochs. The validation loss, however, diverges and the validation accuracy seems unaffected by the training procedure. It is, therefore, likely that the network is currently overfitting to the training data and is not generalising well. Regularisation methods may thus be applied in order to rectify this problem. When the user is satisfied with the training and validation performance of the networks tested using this interface, he or she may enter the values of the hyperparameters found during this manual hyperparameter tuning process, or a small range of values in their vicinity, into the appropriate places in the *deep learning models* tab shown in Figure 7.11 for use during the grid search.

In the final tab of the *develop models* page of the ECCO system, the user can select lexicon-based models to be included in the comparative model evaluation. Since the focus of this dissertation is to demonstrate the working of the ECCO framework with respect to machine learning approaches, in particular, the development of customised lexicon-based methods is not pursued. Off-the-shelf methods were, instead, embedded into the system that do not require

[11] If more than one value is specified for a hyperparameter, the first such value is used to train the model.

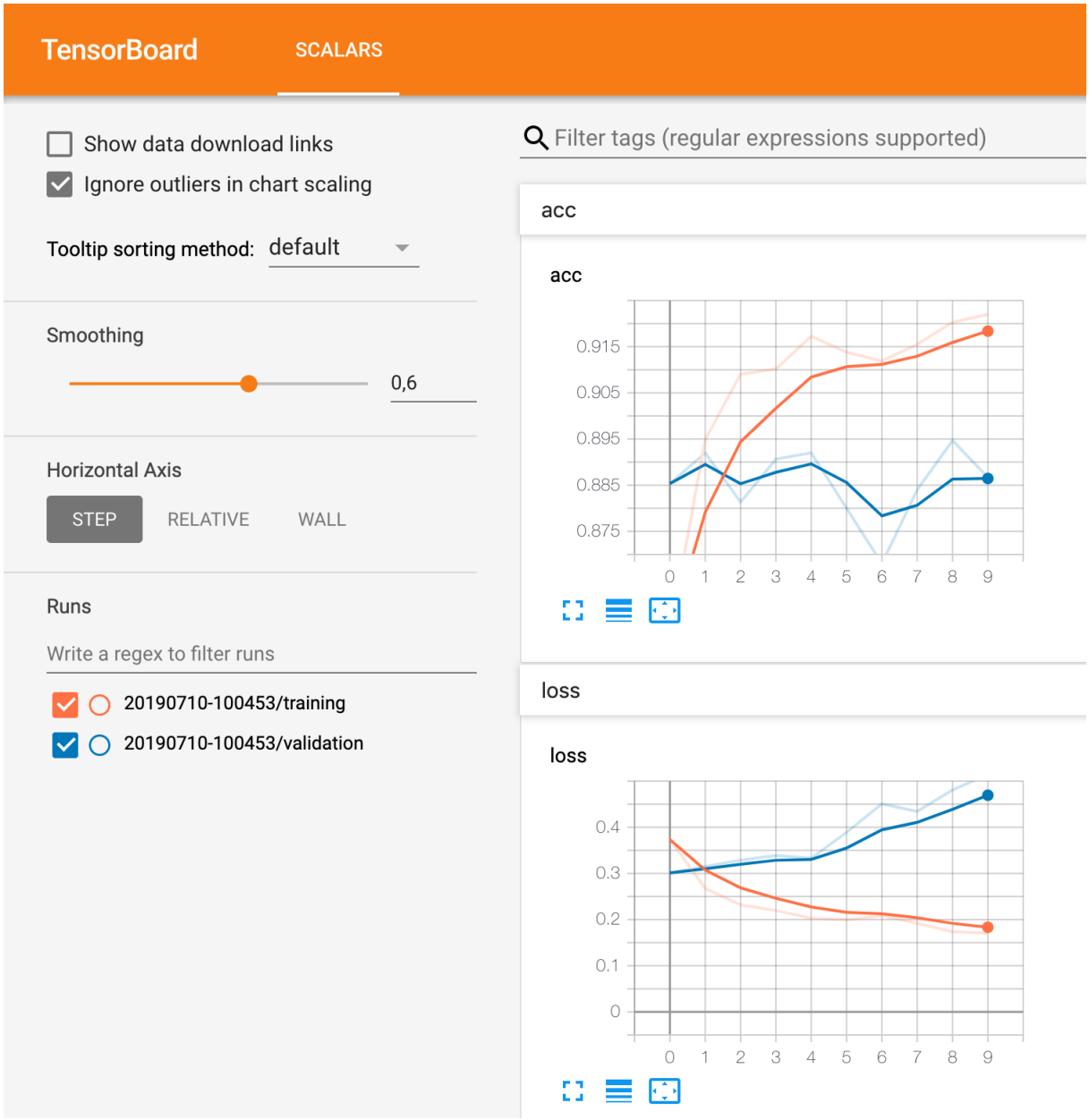


FIGURE 7.12: *The* `Tensorboard` *interface for an ANN model launched to facilitate manual hyperparameter tuning.*

any additional training or configuration. Process 10.0 of the ECCO framework was therefore not implemented in this demonstration. These methods and their usage in the ECCO system were already described with respect to the *select existing model* page of the ECCO system at the beginning of this section. As shown in Figure 7.13, the user can simply select which of these four models to evaluate in respect of the data uploaded to the system in the *lexicon-based models* tab.

Once the user has made the desired selections in the feature engineering and model selection tabs, he or she may proceed to evaluate the models by clicking the *evaluate and compare* button at the bottom right-hand corner of the screen, as shown in Figure 7.13. The ECCO system then retrieves the selections made by the user in each of the tabs and generates *Vectoriser* objects, as described earlier, as well as *parameter grids* specifying the various possible hyperparameter

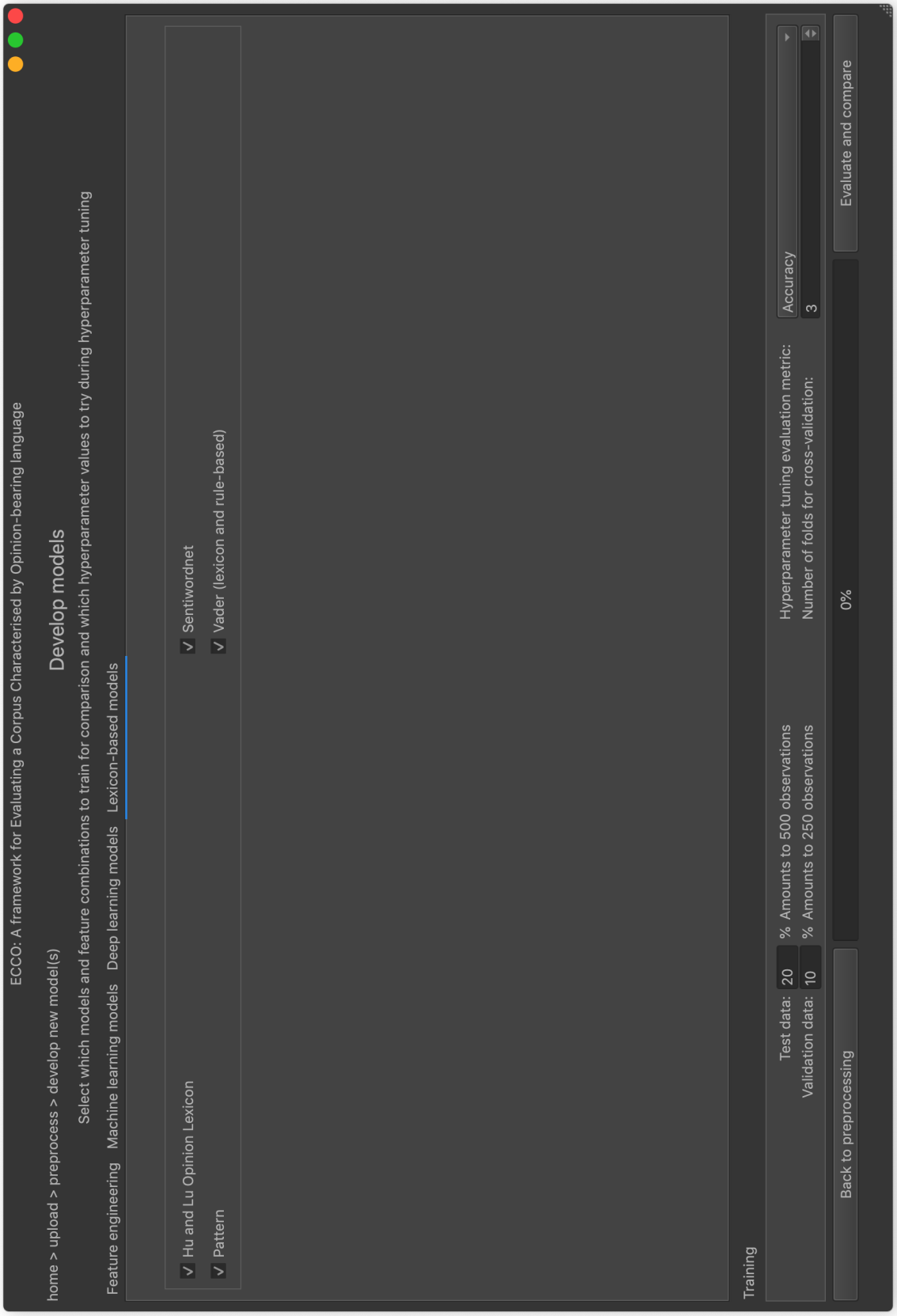


FIGURE 7.13: *The develop models page of the ECCO system with the lexicon-based models tab activate.*

combinations for each of the selected machine learning algorithms. Subsequently, an *Experiment* class is instantiated for each distinct feature-model pair. More specifically, a separate experiment is performed for each instance of the *Vectoriser* class formed according to the specified feature engineering processes in combination with the Naïve Bayes, SVM, logistic regression and ANN models. Given the selection in Figures 7.9–7.11, this results in thirty six experiments (for nine *Vectoriser* class instances and four selected machine learning models). The word embedding feature representations are then combined with the CNN and LSTM models, resulting in a further two experiments. Finally, each of the lexicon-based models also constitutes one experiment, yielding a total of forty two experiments for the selection in Figures 7.9–7.11 and 7.13. Each object of the *Experiment* class is, therefore, optionally composed of an object of the *Model* class and an object of the *Vectoriser* class, as shown in Figure 7.1. The *Model* class, in this case, refers to the class constructed within the ECCO system which may be trained in respect of the labelled *review* data and be deployed to classify documents uploaded into the system once trained. Where applicable, this *Model* object is also composed of an object of the *Estimator* class, which refers to the native classes for machine learning models of the `Scikit-learn` or `Keras` libraries.

Before the models can be evaluated and compared, the hyperparameters of the machine learning models must be tuned. For this purpose, a grid search (see §3.2.1) is initiated over the search space defined by the parameter grid for each algorithm specified by the user on the *develop models* page. The labelled review data are first split into training and test data according to the proportions defined in the *training* tab at the bottom of the page in Figure 7.13. The training data are then partitioned into k folds, where the value of k is also specified by the user in the *training* section. Finally, for each *Experiment* object composed of a *Model* object that represents a machine learning algorithm, and for every unique combination of hyperparameters in the parameter grid for this algorithm, k models are trained, each retaining a different fold of the data as a test set and utilising the remaining folds as a training set. The performance of each of the k resulting models is then evaluated according to the performance metric selected by the user in the *training* section. This metric may be chosen as the model accuracy, precision, recall, F-measure or AUC score (all of which were defined in §3.2.3). The performance achieved in respect of each of the k test sets is then averaged to yield the estimated performance of the model with the given combination of hyperparameters. After having compared this *cross-validated score* for all of the possible hyperparameter combinations for a machine learning algorithm, the model and associated hyperparameters with the highest score is selected by the ECCO system and retrained in respect of the entire training data set. The cross-validated score and the hyperparameters of the selected model are then stored along with the retrained model in the *Model* object. This process is implemented using the `GridSearchCV` class of the `Scikit-learn` library on multiple cores of the computer in order to speed up computation time.

Consider again the example data entered in Figures 7.9–7.11 and 7.13. The parameter grid of the logistic regression algorithm, for example, is given by

$$C : \{0.01, 0.1, 1, 10\},\ \text{solver: } \{\text{'SAG', 'Newton-CG', 'LBFGS'}\},\ \text{max\_iterations: } \{100\}.$$

For each of the *Experiment* objects making use of a logistic regression model, 3-fold cross validation is performed to compare the accuracy of the twelve $(4 \times 3 \times 1)$ distinct hyperparameter combinations defined by this parameter grid, and the model achieving the highest cross-validated accuracy is selected for each experiment.

As was explained in §3.2.2, however, this cross-validated score cannot be employed to evaluate and compare several models, since the data in respect of which the score is calculated were used to tune the hyperparameters and thus do not constitute *unseen data*. As is common in the

machine learning community, the performance of the models is therefore evaluated in respect of a separate *hold out set* or test set. In the ECCO system, the proportion of the data that constitute this test set is specified by the user in the *training* section. In the example in Figure 7.13, this proportion was set as 20% of the data, amounting to 500 observations. Each of the *Experiment* objects is then evaluated in respect of these data in terms of accuracy, precision, recall, F-measure and AUC score. Apart from the model accuracy, each of these metrics is defined in the context of a binary classification problem. In order to evaluate performance in respect of a multi-class classification problem (*e.g.* in terms of the three sentiment classes *positive*, *negative* and *neutral*), these metrics were therefore calculated for each of the sentiment classes and then averaged. Consider the confusion matrix in Table 7.3 illustrating the prediction results of a model.

	Prediction negative	Prediction neutral	Prediction negative
Condition negative	342	18	4
Condition neutral	36	38	6
Condition positive	4	3	49

TABLE 7.3: *Confusion matrix illustrating the classification results of a model in respect of a testing data set of 500 observations.*

From §3.2.3, the accuracy of this model is calculated as the proportion of correctly classified observations, or in this example as $(342 + 38 + 49)/(342 + 18 + 4 + 36 + 38 + 6 + 4 + 3 + 49) = 429/500 = 0.8580$. The recall for each class is defined as the ratio of the number of correctly classified observations in the class to the number of observations that do, in fact, belong to this class, or $TP/P = TP/(TP + FN)$. In the given example, the precision for the negative class is $342/(342 + 18 + 4) = 342/364 = 0.9396$. Similarly, the precision score for the neutral and positive classes may be computed as $38/(3638 + 6) = 38/80 = 0.4750$ and $49/(4+3+49) = 49/56 = 0.8750$, respectively. Several approaches may be taken to compute the average of these scores, including *macro averaging* and the *micro averaging*. The macro average is the simple arithmetic mean of the class scores. In the example above, the macro average precision is therefore computed as $(0.93964 + 0.4750 + 0.8752)/3 = 0.7632$. When adopting this approach, each class is weighted equally, irrespective of the number of observations in the class. If a particularly large or a particularly small score is achieved in an under-represented class, this may skew results considerably. Instead, one may compute the micro average, in which case the score is computed globally. For a three-class classification problem with classes 1, 2 and 3, the micro-averaged precision score is given by [185]

$$precision_{micro} = \frac{TP_1 + TP_2 + TP_3}{P_1 + P_2 + P_3}, \tag{7.1}$$

where P_i denotes the number of observations that belong to class i and TP_i denotes the number of correctly classified observations in class i. In the example in Table 7.3, this yields $(342 + 38 + 49)/((342 + 18 + 4) + (36 + 38 + 6) + (4 + 3 + 49)) = 429/500 = 0.8580$, which is equal to the accuracy. As it turns out, micro-averaged precision, recall and F-measure scores in a multi-class setting are mathematically equivalent to the accuracy score [185]. It is typically suggested that micro-averaging is used when a class-imbalance is suspected [249] since this gives equal weight to each *observation* in the data set, rather than to each *class*. Since the objective of a sentiment polarity classification model is typically to correctly classify as many cases as possible, and since class imbalance may well be a problem in this context, micro-averaging is employed for these metrics in the ECCO system.

For the calculation of the AUC score, a similar approach was adopted. Since multi-class AUC scores are not supported by the `Scikit-learn` library, however, a workaround was implemented. More specifically, the AUC scores are obtained for each class by means of a *one-versus-all* approach and these scores are then aggregated by means of a *weighted average* according to the number of observations in each class, approximating a micro average. For scoring and probabilistic classifiers (all classifiers except for SVM and the lexicon-based models), the numerical scores for each class are employed when computing the AUC score. For each class, the binary problem is considered where each observation is classified as belonging to that class (a *positive* observation) or not belonging to that class (a *negative* observation). The score assigned to that class by the classifier is then taken as the *positive* probability. Consider the example in Table 7.4, where four observations are assigned scores for the positive, negative and neutral class by a scoring classifier.

Observation	True class	Positive score	Neutral score	Negative score	Predicted class
1	positive	0.7	0.2	0.1	positive
2	negative	0.2	0.5	0.3	neutral
3	negative	0.1	0.1	0.8	negative
4	neutral	0.1	0.5	0.4	neutral

TABLE 7.4: *A hypothetical example of the output of a scoring classifier for a data set with four observations.*

In order to calculate the AUC score for the negative class, the problem is converted to the binary problem in which an observation is classified as belonging to the negative class (indicated by a 1) or not belonging to the negative class (indicated by a 0). The transformed problem is shown in Table 7.5. As described in §3.2.3, the ROC curve can then be constructed by varying the threshold used to classify an observation. A threshold of 0.4, for example, would result in observations 3 and 4 being classified as class 1 and observations 1 and 2 being classified in class 0, yielding a true positive rate of $1/2 = 0.5$ and a false positive rate of $1/1 = 0.5$.

Observation	True class	Probability
1	0	0.1
2	1	0.3
3	1	0.8
4	0	0.4

TABLE 7.5: *The binary problem considered when computing the AUC score for the negative class based on the output in Table 7.4.*

For a discrete classifier predicting the classes in the last column of Table 7.4 directly (without the scores for each class), the probabilities in Table 7.5 would be set to 0, 0, 1 and 0 for the four observations, respectively. At any threshold other than 1 or 0 this results in a true positive rate $TPR = 1/2 = 0.5$ and a false positive rate $FPR = 0/2 = 0$. The ROC curve for the discrete classifier is then estimated by linearly interpolating between the points (0,0), (TPR, FPR), and (0,1). The difference between the resulting ROC curves for the discrete and scoring classifiers is shown in Figure 7.14. Due to the limited number of observations in the sample, the ROC curve for the scoring classifier has only two *steps*. This number increases with the number of observations, resulting in a graph more similar to the traditional *curve* shown in Figure 3.5.

A similar procedure is followed to construct the ROC curves for the positive and neutral classes. After having computed the areas under each of these curves, the aggregated AUC score is then

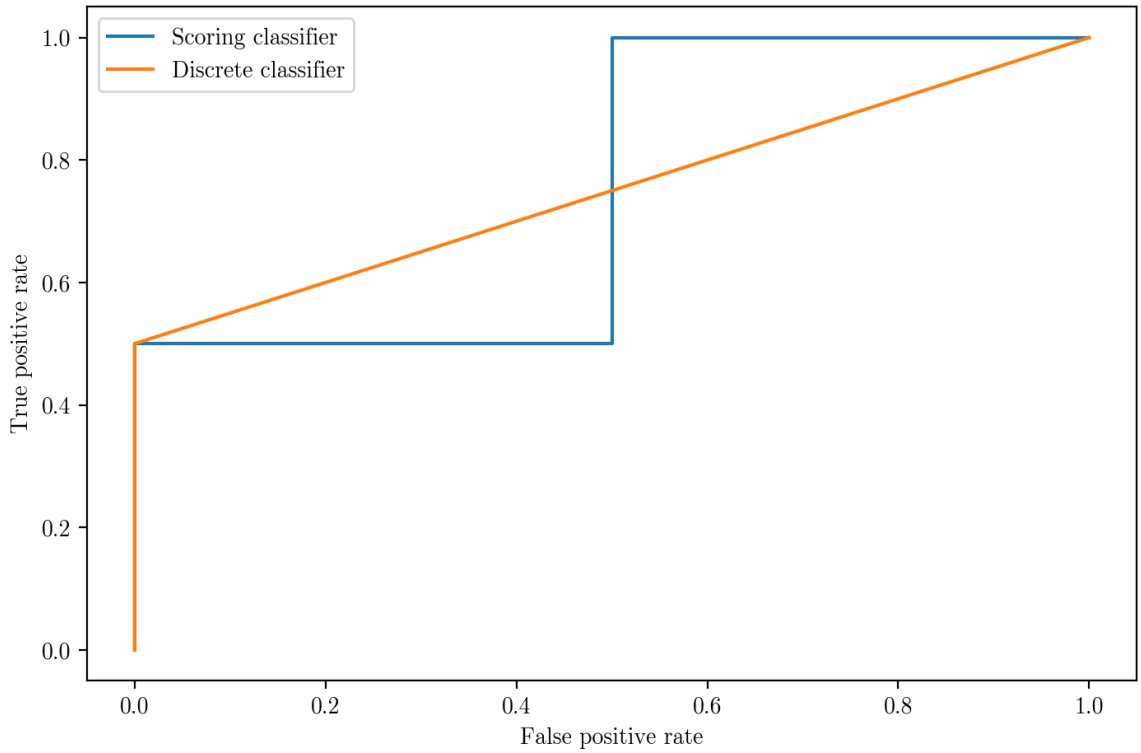


FIGURE 7.14: *The ROC curves for a scoring classifier and a discrete classifier in the case of the negative class of the example in Table 7.14.*

computed as the weighted average of the AUC scores for each class, where the weight for each class is determined as the ratio of the number of observations in that class to the total number of observations.

The scores achieved by the models in each experiment are subsequently saved as attributes of the associated *Experiment* object, and the contents of each object are stored on the user's computer under the *log* directory in the system folder for future reference. Thereafter, the results of the experiments are presented to the user in the form of a *table* widget on the *evaluate models* page of the ECCO system, the final heading under the *model* category in Figure 7.2. As shown in Figure 7.15, each experiment forms a row in this table, with the column entries corresponding to the experiment number, the model or algorithm used, the document representation and n-gram range employed as features, where applicable, as well as the method of sampling by which the training, validation and test data sets were generated from the original data set[12] and, finally, the scores for each of the metrics. The user is able to sort the table based on any of the columns. In the figure, the table is sorted according to decreasing model accuracy, revealing the top performing models to be logistic regression, ANN and CNN with accuracies of 0.8540, 0.8540 and 0.8520, respectively. Furthermore, upon clicking on any of the entries in the table, the user is provided with more information on the model and its classification performance. In particular, the hyperparameters selected during the grid search are shown for the machine learning models, along with the score achieved (in respect of the metric selected by the user) during cross-validation in the bottom left-hand corner of the screen. Moreover, a confusion matrix containing the classification results in respect of the test data set is shown in the bottom

[12]The default sampling method employed in the ECCO system is *stratified* sampling, where the data are partitioned into training and test data sets that have approximately the same class distribution as the original data set.

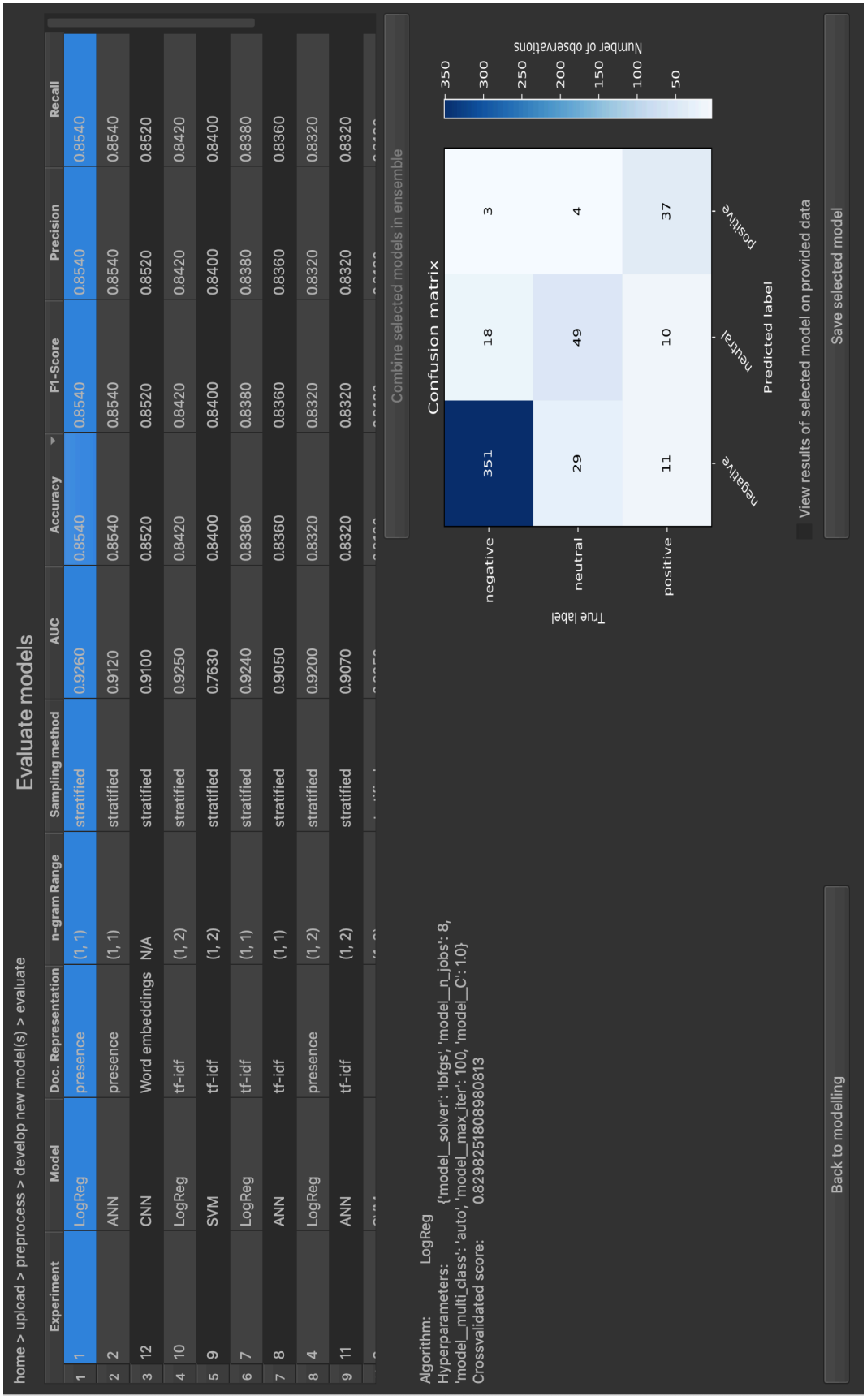


FIGURE 7.15: *The evaluate models tab of the develop models page.*

right-hand corner of the screen. The selected model in the figure is logistic regression. For this model, the variable hyperparameters entered by the user in Figure 7.10 were the inverse regularisation strength C and the *solver* algorithm employed to optimise model parameters. Upon examining the output on the bottom left-hand side of the screen, it is clear that $C = 1$ and the LBFGS algorithm were selected by the system, upon which the model achieved a cross-validated accuracy score of 0.8298. The entries of the confusion matrix are shaded according to the number of observations in each category, where a larger number of observations results in a darker shade of blue. It can thereby be determined at a glance that the model was primarily *'confused'* between negative and neutral observations. This interface thus enables the user to determine which models have performed best in respect of various metrics, and to elicit more detailed information about selected models and their shortcomings.

If more than one experiment (row) is selected in the table, the *combine selected models in ensemble* button is enabled. Upon clicking this button, the *combine models* dialogue window is launched, by means of which the user can configure an appropriate ensemble model with the selected experiments in the table as base learners. More specifically, the combination method may be configured to employ either discrete outputs of the base learners, or their scoring (or probabilistic) outputs if these are available, as shown in Figure 7.16. Furthermore, these outputs may be combined by means of simple plurality voting, weighted voting or a meta-learning approach (each of which were described in §3.4). In the weighted voting approach, the weight w_i of each base learner $i \in \{1, \ldots, K\}$ is

$$w_i = \frac{\alpha_i}{\sum_{j=1}^{K} \alpha_j}$$

as per Opitz and Shavlik [223], where α_i is the cross-validated score achieved during the grid search for machine learning models, and the accuracy in respect of the training data for lexicon-based models. If the meta-learning approach is selected, the algorithm employed by the meta-learner may, furthermore, be specified by means of a dropdown list. In this particular instantiation of the ECCO framework, however, only the most popular meta-learning algorithm, namely logistic regression, has been implemented as a proof of concept.

The simple and weighted voting ensembles are implemented *via* the `VotingClassifier` class of the `Scikit-learn` library [236], whilst the meta-learning approach is facilitated by means of the `StackingClassifier` class of the same library. In the stacked ensemble, the training data for the meta-learner are generated by means of five-fold cross-validation in respect of the training data according to the process described in §3.4. For the classification of new data points, the base learners are then re-trained in respect of the entire training data set. If scoring outputs are employed as input to the meta-learning algorithm, the scores or probabilities for all classes are employed (unlike in the StackingC approach reviewed in §3.4). In the case of binary classification, however, only one class is considered, since the probabilities of Class 0 and Class 1 are perfect predictors of one another by means of the relationship $p_1 = 1 - p_0$. Finally, for the sake of simplicity, default hyperparameters are employed in the meta-learning algorithm. More specifically, the LBFGS optimisation algorithm is employed to determine model parameters within a limit of 100 iterations, and the inverse regularisation strength C is equal to 1.0.

Upon clicking the *OK* button in the dialogue window, the ensemble model is constructed according to the selected configuration and trained, where applicable, in respect of the training data. The model is then evaluated according to the same process employed for the previous models and an *Experiment* class is instantiated for the ensemble model. Finally, the ensemble is added to the table widget shown in Figure 7.15 and may be treated like any other experiment by the user. Both *competitive* model selection (selecting the best or most appropriate model)

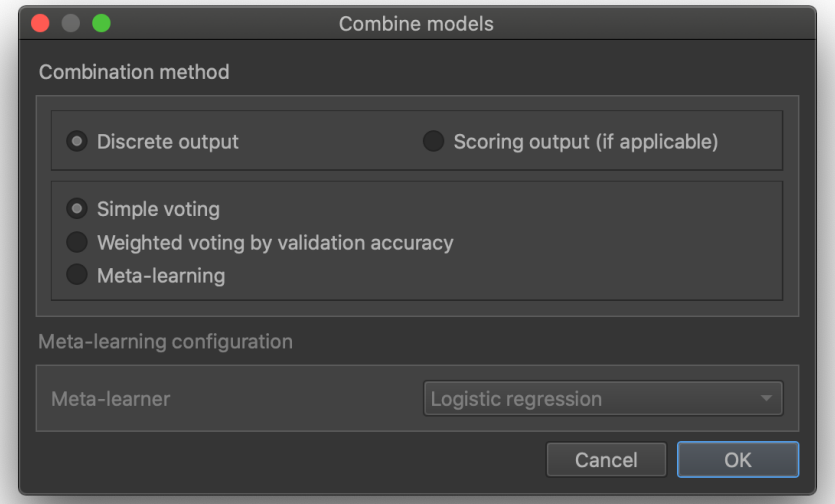


FIGURE 7.16: *The ensemble learning dialogue window.*

and *complementary* model selection (combining several models to form an ensemble model) are therefore accommodated in the ECCO system.

Any number of models can then be saved using the *save selected model* button below the confusion matrix. These models can then be retrieved at a later stage *via* the *select existing model* tab of the system, as previously described. All models are saved as objects in a `JSON` file format that take a corpus of documents as input and return *label predictions* (*positive*, *negative* or *neutral*) for each document, as well as *numerical predictions* (the probability or score that a document belongs to each class) for each document, where applicable. Machine learning models are thus stored together with their associated *Vectoriser* objects in a `Pipeline` object, where applicable. By checking the *view results of selected model on provided data* check box above this button before clicking it, the user can deploy the selected model in respect of the data uploaded into the ECCO system and analyse the model results *via* the *Dashboard* interface subsequently launched, as is described in the following section.

7.2.3 Implementation of the analysis component

Once a model[13] has been selected to initiate the analysis phase, the sentiment polarity of each of the documents uploaded by the user prior to the preprocessing phase is classified as either *positive*, *negative* or *neutral* by this model (Process 13.0 of the ECCO framework). The *reviews* data and the *supplementary data* set are then merged[14] along with these model results to yield the *results* data set for analysis by means of the *Dashboard* interface. Subsequently, this interface is launched in the user's default browser.

[13]The term *model* is henceforth used to refer to lexicon-based models, as well as machine learning models combined with their associated *Vectoriser* objects or word embedding layers.

[14]An *inner* merge is employed for this purpose. Consequently, if any entries in the reviews data set are not linked with a corresponding entry in the supplementary data set, such entries are deleted.

As is shown in Figure 7.17, the dashboard is partitioned into five tabs, each of which relate to one of the five headings under the *analyse* category in Figure 7.2. The first two tabs facilitate the analysis of the corpus and its associated sentiment distribution without regard for the additional data provided by the user, whilst the remaining three tabs are aimed at analysing the relationship between a document's sentiment class and these additional structured data.

The active tab in Figure 7.17 is the *summary view* tab, where the user is presented with *traditional* summaries of the corpus (Process 16.0 in the ECCO framework). More specifically, a pie chart illustrating the distribution of the sentiment polarities in the corpus is shown on the left-hand side of the page. In the example in the figure, 75.10% of the data were classified as *negative* by the chosen model, whilst 15.50% and 9.39% were classified as *neutral* and *positive*, respectively. All the graphs in the *Dashboard* interface are interactive. By clicking on the entries in the legend in the top right-hand corner of the plot, for instance, the user can isolate certain classes in the graph. Furthermore, upon hovering the mouse over any graph, the user is provided with more information on the visualised data. In the case of the pie chart, for example, the raw count of each sentiment class is given when the user hovers the mouse over the associated portion of the chart.

On the right-hand side of the page, a textual summary is given. In this case, template instantiation (see §4.2.4) is used to inform the reader of the total number of documents analysed and the number of documents classified into each sentiment class, as well as the top keywords occurring in documents of each class. These keywords are extracted by means of the function for POS tagging from the `NLTK` library. More specifically, the nouns most frequently occurring in the documents of the positive and negative sentiment classes are extracted. In the example in the figure, the top three keywords for the positive class are *service*, *sorry* and *everything*, whilst the top three keywords for the negative class are *money*, *loan* and *bank*. It should be noted that the noun extraction algorithm may mistakenly extract keywords that are not nouns in cases of poor sentence structure, since parse trees are used to generate the POS tags as described in §4.1.2.

Upon *scrolling* down on the *summary view* tab, the user can view randomly selected samples of the original review documents assigned to each of the sentiment classes, as shown in Figure 7.18. This allows the user to gain more detailed insight into the types of responses associated with each class, as well as their topical content. New random samples are drawn from the corpus whenever the *randomise samples* button is clicked. In the figure, for example, the sampled negative review pertains to a department not following through on a promise to contact a customer, whilst the sampled positive review commends good service. The sampled neutral review constitutes a one-word answer, which is difficult to contextualise. Furthermore, an overview of the content of the documents in each sentiment class is provided in the form of a word cloud, similar to that displayed during the preprocessing phase in Figures 7.5 and 7.6. From the word clouds shown in the figure it is, for example, clear that *loan*, *bank*, *money*, *account* and *ATM* are frequent keywords in negative reviews, whilst positive reviews typically mention *good* or *great service*. Once again, the most prevalent words in neutral reviews, namely *ok* and *need*, are not rich in information. Certain words may be excluded from the word clouds by typing these into the input field at the bottom of the page. In this manner, the focus on particularly frequently used words can be reduced in order to reveal other important terms that are prevalent in the relevant documents.

In order to gain a deeper insight into the topics discussed in the corpus, the *topic analysis* tab may be employed. Two approaches are facilitated by the ECCO system in this case, namely noun phrase detection and LDA topic modelling, as shown in Figure 7.19. Both of these approaches facilitate Process 15.1 in the ECCO framework pertaining to topic extraction. In the former approach, the `NLTK` library is used to parse each sentence in the corpus in order to extract noun

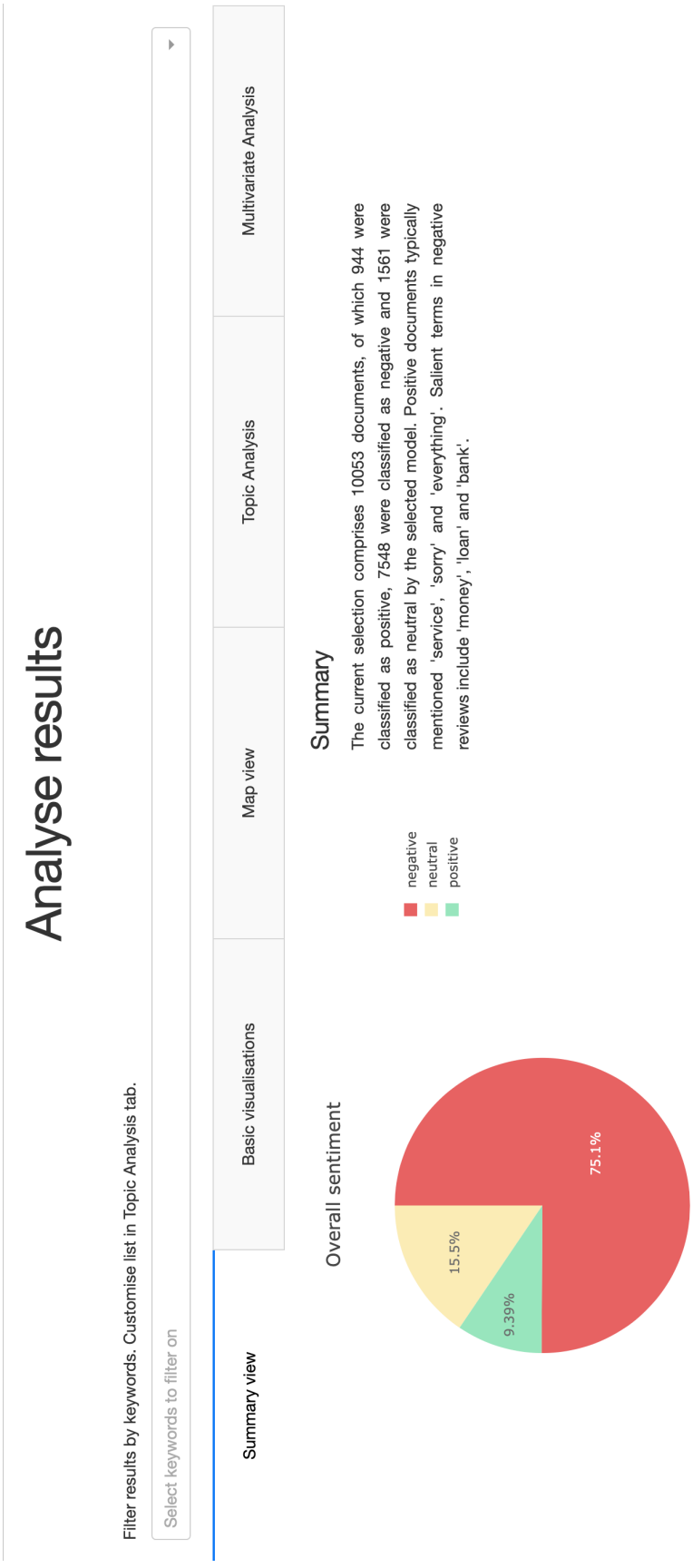


FIGURE 7.17: *The summary view tab of the Dashboard interface. A pie chart on the left-hand side visualises the distribution of the sentiment classes in the corpus, and a textual summary is given on the right-hand side.*

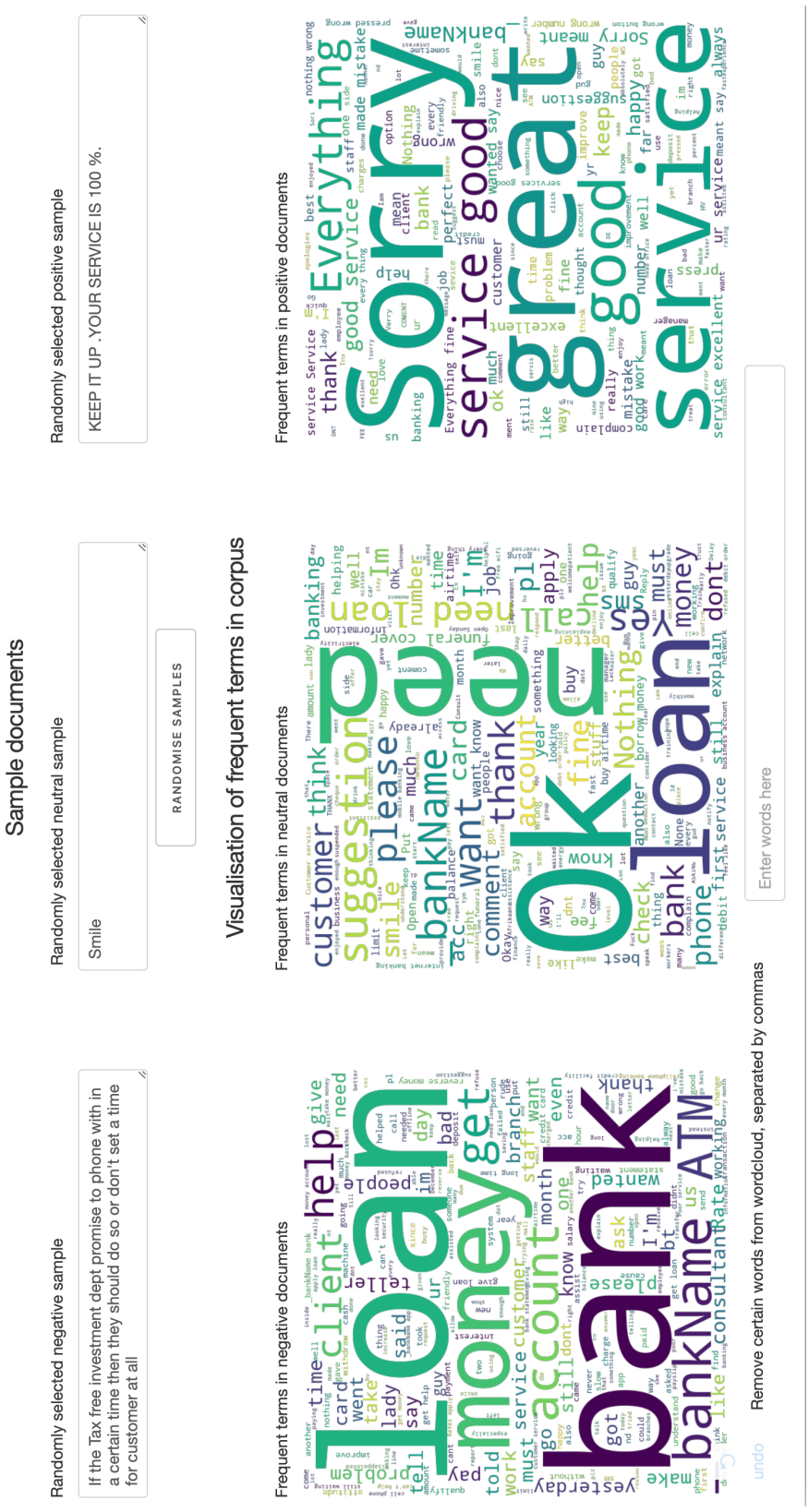


FIGURE 7.18: *The summary view tab of the Dashboard interface (continued). Reviews from each sentiment class can be sampled randomly in the top section, whilst word clouds are displayed to provide an overview of the most frequently used terms in reviews of each sentiment class.*

phrases, as was done for the textual summary in the *summary view* tab. A drop-down list is then populated with the thirty most frequently used noun phrases. The user may amend this list by selecting the relevant nouns from this drop down list and by adding new phrases to the list *via* the input field and the associated *add* button. In the example in Figure 7.19, the keywords *loan, service, atm, credit, staff, rates, app* and *card* were selected by the user.

In the latter approach, an LDA topic model (see §2.3.3) is fitted to the data according to the number of topics specified by the user. The *bag of words* representation of the review documents forms the input to this model, where only terms occurring in at least two documents and no more than 50% of the documents are retained in an attempt to filter out common, uninformative words. The `lda` package from the `gensim` library [256] is then used to fit an LDA model to this representation. The *online Variational Bayes* algorithm used to train the model is detailed in Hoffman *et al.* [125]. By default, one *pass* or iteration is made through the entire corpus during training. If a more precise result is required by the user, this number can be increased using the *number of iterations* input field. Upon clicking the *execute and view topic model* button, the user is redirected to a new tab in his or her browser, where the topic model is visualised. For this purpose, the visualisation package *LDAvis* by Sievert and Shirley [287] was implemented, which takes as input the following outputs generated by LDA [286]:

(i) The $K \times W$ matrix $\boldsymbol{\phi}$, containing the estimated probability mass function over the W terms in the vocabulary for each of the K topics in the model,

(ii) the $D \times K$ matrix $\boldsymbol{\theta}$, containing the estimated probability mass function over the K topics in the model for each of the D documents in the corpus,

(iii) the number n_d of tokens observed in document $d \in \{1, \ldots, D\}$,

(iv) the character vector of length W containing the terms in the vocabulary, and

(v) the frequency M_w of term w across the entire corpus for each term $w \in \{1, \ldots, W\}$.

The resulting visualisation for the example data with $K = 5$ topics and two iterations is shown in Figure 7.20. In the left-hand plot, each of the topics is represented by a circle in a two-dimensional plot. The centres of the circles are determined by computing the distance[15] between the two topics according to the term-topic distributions learnt during the training of the LDA model. This distance is scaled to two dimensions by application of PCA (see §2.3.1). The area of each of the circles is proportional to $N_k / \sum_k N_k$, where $N_k = \sum_{d=1}^{D} \theta_{dk} n_d$ is the estimated number of tokens that were generated by topic k across the entire corpus. If the user hovers the mouse over one of the terms displayed on the right-hand side of the screen, the areas of the circles are changed to be proportional to $P_{kw} / \sum_k P_{kw}$ instead, where $P_{kw} = \phi_{kw} N_k$ is an estimate of the frequency with which term w was generated by topic K.

If no topic is selected, the terms on the right-hand side of the screen represent the thirty most *salient* terms in the corpus, where saliency is calculated according to the formula developed by Chuang *et al.* [53] as

$$s(w) = p_w \sum_K P(k \mid w) \log \left(\frac{P(k \mid w)}{p_w} \right),$$

where $p_w = M_w / \sum_w M_w$ and $P(k \mid w) = \phi_{kw} / \sum_k \phi_{kw}$. The width of the bar next to each term, which is coloured in blue, is then set to $\sum_k P_{kw}$, the estimated total number of occurrences of

[15] In this case, *Jensen-Shannon divergence* is adopted as a measure of the similarity between the probability distributions contained in $\boldsymbol{\phi}$. Details of this method may be found in [91].

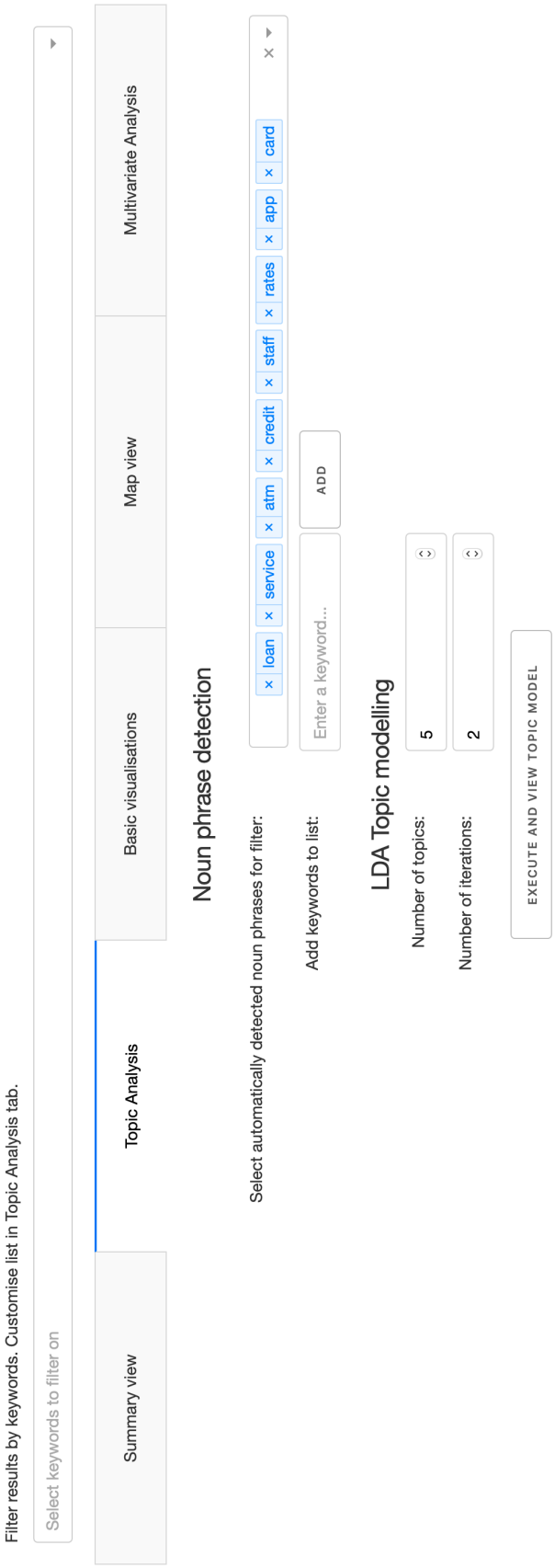


FIGURE 7.19: *The topic analysis tab of the Dashboard interface. Both noun phrase detection and LDA topic modelling are facilitated.*

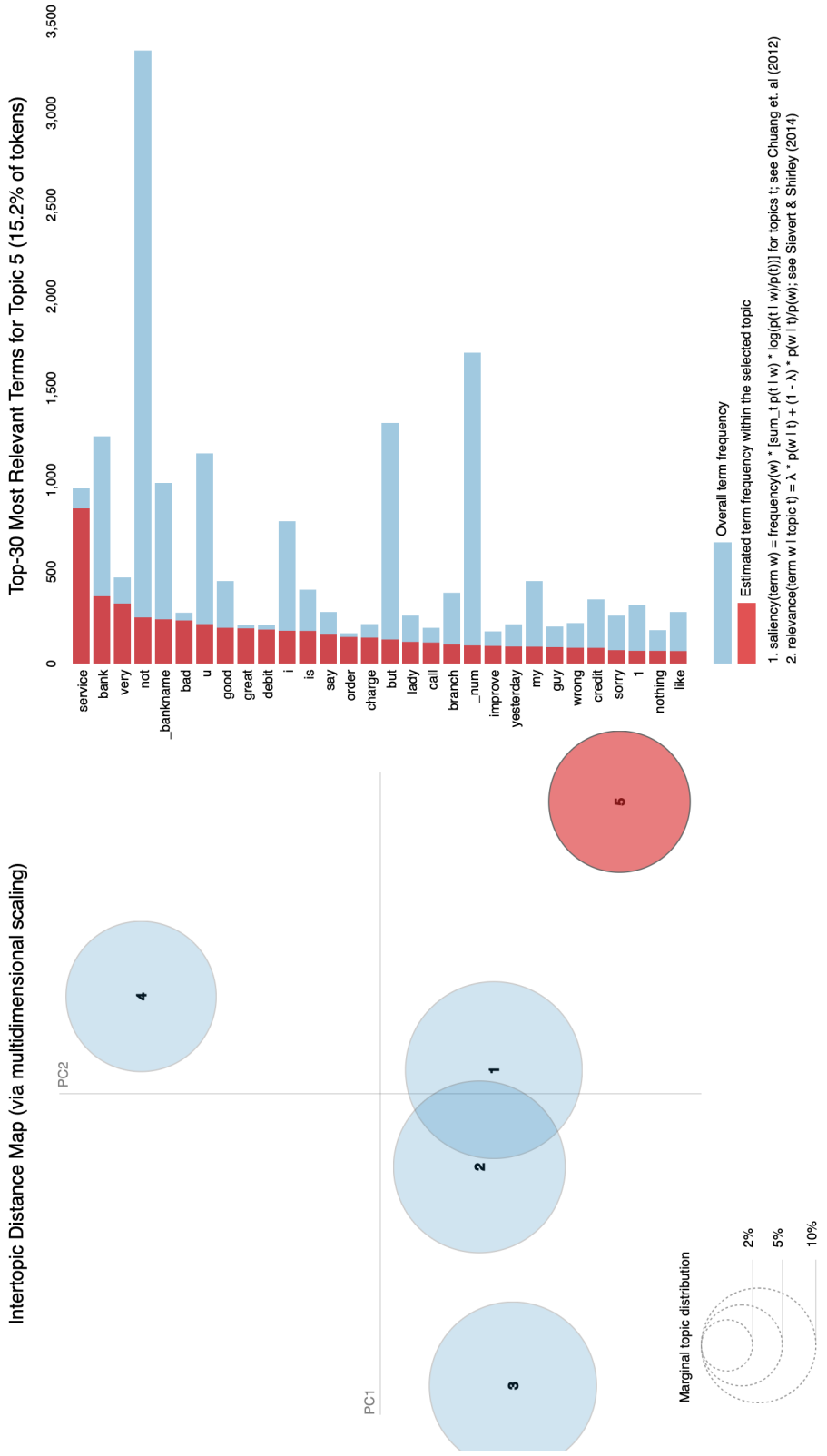


FIGURE 7.20: *The LDAvis interface launched to analyse the results of the LDA topic model. In this case, Topic 5 is selected and the most relevant terms to this topic are shown on the right-hand side of the graph, where blue bars represent the saliency of these words in the corpus in general and red bars represent the relevance of these terms to Topic 5 in particular.*

term w in the corpus. If, however, a topic is selected, the terms are ordered according to their *relevance* to a particular topic, where the relevance of a term w to topic k is calculated as [287]

$$r(w, k \mid \lambda) = \lambda \log(\phi_{kw}) + (1 - \lambda) \log \left(\frac{\phi_{kw}}{p_w} \right),$$

where λ is a user-specified parameter that can be adjusted using the *slider* in the top right-hand corner of the page as shown in Figure 7.20. The width of the red bar that is subsequently overlaid on the blue bar next to each term is set to P_{kw}. Upon exploring the topics and important terms in the corpus, the user can therefore gain insight into the frequency with which these topics and terms were present in the data. With reference to the example in Figure 7.20, for instance, it is clear that the five topics are relatively well separated with the exception of topics 1 and 2, which exhibit regions of overlap. The most relevant terms for Topic 5 include *service*, *bank* and *very*. Furthermore, upon comparing the width of the red and blue bars, one can identify terms which appear almost exclusively within the selected topic, such as *service*, *bad*, and *great*. The relevant terms identified using the *LDAvis* interface may then be inserted into the list of noun phrases or keywords in the *topic analysis* tab in Figure 7.19.

These keywords may be applied to further analyse the results of the sentiment classification model in two ways. First, any number of keywords may be selected in the drop-down list above the tab headers fitted with the label *filter results by keywords [. . .]*. This constitutes the filtering process in the ECCO framework (Process 14.0) and allows the user to execute the functions in tabs 1, 3, 4 and 5 using only that subset of the documents which contain these keywords. Secondly, at the bottom of the *topic analysis* tab, the sentiment counts for specific keywords may be visualised and compared by means of a histogram (Process 15.2 of the ECCO framework). The vertical axis in this plot represents the number of documents in each sentiment class that contain the keywords plotted on the horizontal axis, as shown in Figure 7.21.

The user can dynamically compare any selected number of topics by altering the selection in the drop-down list above the graph. Furthermore, the user may isolate any of the sentiment classes by clicking on their respective entries in the legend at the top right-hand corner of the plot. By selecting the appropriate setting, using the *radio buttons* in the top left-hand corner of the plot, the user may also change the units of the vertical axis from *raw counts* (the number of documents in each sentiment class that contain the keywords) to *normalised counts* (the proportion of documents in each sentiment class that contain the keywords).

From the example in Figure 7.21, it is clear that the vast majority of documents mentioning the words *loan*, *rates* or *atm* were classified as *negative*, whilst almost half the documents that mention *service* were classified as *positive*. Considering the results in Figure 7.20, this seems to make sense, with the rather isolated Topic 5 containing service as one of its most relevant words, along with several adjectives such as *great* and *bad*. Furthermore, given the current selection of keywords in Figure 7.21, it would seem that over 20% of *negative* reviews mention *money* or *loan*.

Whereas the *summary view* tab provides an overview of the content and sentiment distribution of the *unstructured* component of the user's data, and the topic analysis tab provides an in-depth view into the content of the corpus, the *basic visualisations* tab analyses the relationship between the *structured* data attributes and the sentiment polarity of documents. As shown in Figure 7.22, this includes stacked histograms or *count plots* for qualitative variables (on the left-hand side of the page) and box plots for quantitative variables (on the right-hand side of the page). In the former case, the user can select one of the qualitative data attributes (uploaded for either the review data set or the supplementary data set) from the drop-down list. A stacked

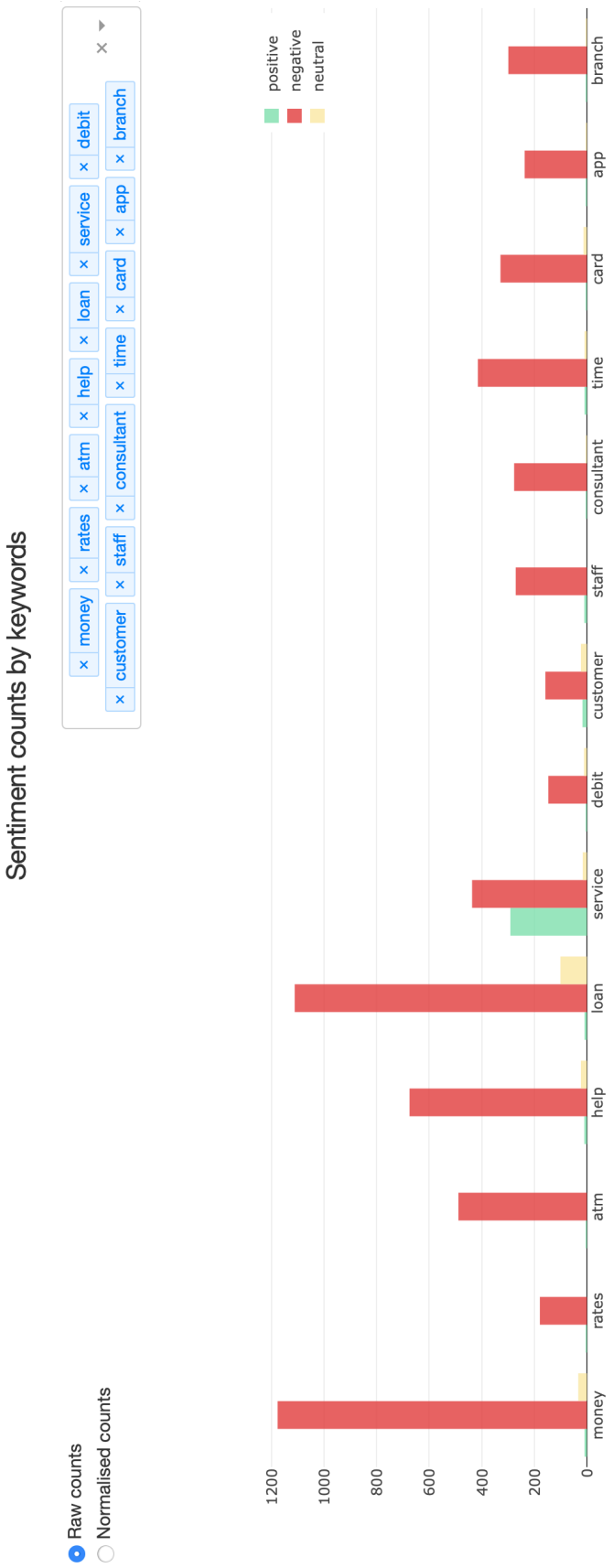


FIGURE 7.21: *The topic analysis tab of the Dashboard interface (continued). The number of documents of each sentiment class that contain the keywords on the horizontal axis is visualised by means of a histogram or count plot.*

histogram is then displayed for each of the possible values of this variable, illustrating the number of documents assigned to each of the sentiment classes in each case.

In the example in the figure, for instance, the number of *positive*, *negative* and *neutral* documents is shown, segmented by the *primary need* associated with each document entry. Most documents in this example were associated with the primary need to *transact*. In §4.2.4, the disadvantages of using a stacked histogram for illustrative purposes are discussed with reference to the difficulty of interpreting such a graph. Liu *et al.* [177] proposed plotting positive and negative sentiment counts on either side of the horizontal axis in order to alleviate this problem. Since all graphs in the dashboard interface of the ECCO system are interactive, however, this was not necessary. The user can instead isolate the trace of any of the sentiment counts for easy comparison. Furthermore, the user is able to change the vertical axis of the graph from displaying *raw counts* to *normalised counts* representing the proportion of documents in each category belonging to each of the three sentiment classes. As before, this is facilitated by the *radio buttons* next to the drop-down list. By presenting the user with both representations of the same graph, the distribution of sentiment in each category may be compared whilst retaining sensitivity to the amount of observations in each category (distributions for very small sample sizes should be interpreted with caution).

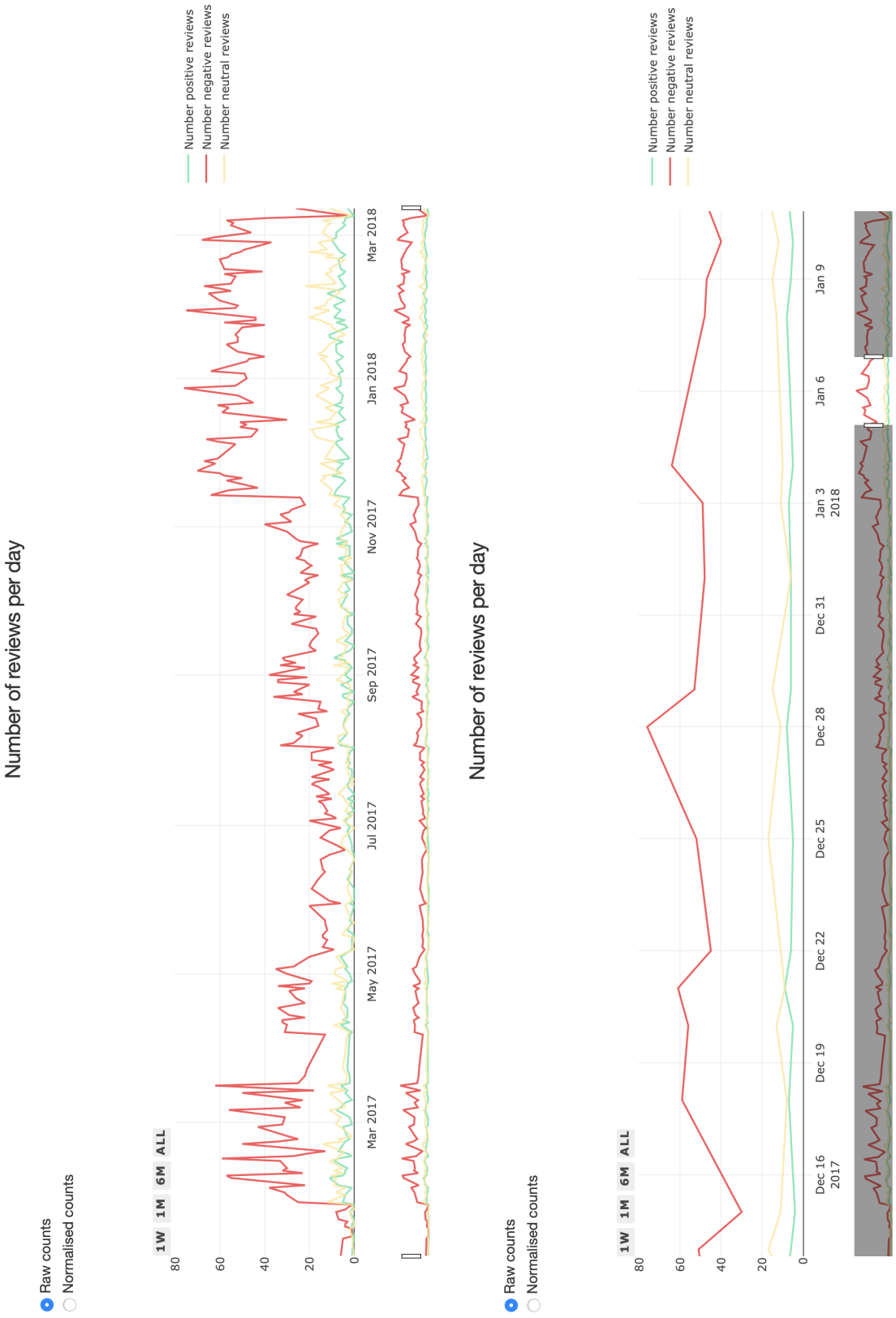
The box plots on the right-hand side of the page show the distribution of the values of quantitative variables in respect of the sentiment class. The example in the figure shows that the variable *Average_Monthly_Fee* has approximately the same interquartile range for all three sentiment classes, but that the outlier values are significantly larger (in the negative) for the *negative* sentiment class than for the *positive* and *neutral* classes. Upon hovering the mouse over any of the box plots, the user is provided with a summary of the values used to generate these plots. As shown in the figure, these include the value of the 25^{th}, 50^{th} and 75^{th} percentiles of the data, as well as the endpoints of the *whiskers* of the box plot (*lower fence* and *upper fence*) and the minimum and maximum values of the data. Furthermore, (s)he is able to isolate any of the sentiment classes, as before.

In the final graph of the *basic visualisations* page, the *distinct* variable denoting the date on which a review was received is explored. A time series graph is constructed in which the horizontal axis represents this date variable and the vertical axis represents the number of reviews received on each date. As is shown in Figure 7.23, a separate line is plotted for each sentiment class. In this manner, it may be elucidated when a particularly large number of reviews was received for a particular sentiment class, possibly indicating the occurrence of a positive or negative triggering event. The user may also choose to plot normalised counts instead, in order to separate such cases (that may have been triggered by bad publicity, for example) from cases where a large number of total reviews was received (perhaps due to a call for action to customers) by evaluating whether the *proportion* of one sentiment class has increased. Finally, the user may also *zoom in* on a specific date range *via* the *window slider* underneath the graph, as shown in Figure 7.23(b). The window size for this slider may be set to one of the pre-specified period lengths determined by the buttons in the top left-hand corner of the graph (where *1W* denotes one week, *1M* denotes one month and *6M* denotes a six-month period), or may be adjusted manually.

From the example in Figure 7.23, it is clear that primarily negative reviews were received throughout the analysis period, with the frequency increasing to between 40 and 60 negative reviews per day between November 2017 and March 2018. The number of neutral and positive reviews received per day is comparatively small, but also increases towards the end of the period.

In the fourth tab of the Dashboard interface, the *map view*, the relationship between the distribution of sentiment and the second *distinct* variable is explored: Location. The latitude and longitude values categorised as such by the user during the preprocessing phase are employed to

FIGURE 7.22: *The basic visualisations tab of the Dashboard interface. Histograms or count plots are shown for qualitative variables (left), whilst box plots are shown for quantitative variables (right).*



(a) *Time series plot with full range of dates* (b) *Time series plot with a time window of one month selected*

FIGURE 7.23: *The basic visualisations tab of the Dashboard interface (continued). The time series plot displaying the number of reviews of each sentiment class in (a) can be zoomed in on to display a custom range of dates as in (b).*

generate a so-called *bubble map*, as shown in Figure 7.24. Three concentric circles are plotted for each latitude and longitude value associated with a document. Each of these circles represents the number of documents associated with this location that were classified as *positive*, *negative* and *neutral*, respectively. If the number of documents in a particular sentiment class outweighs the other sentiment classes at a given location, the associated colour will dominate the visualisation. If a *'hot spot'* of negative reviews were present in a particular location, for example, this would be indicated on the map as a particularly large and red region.

In the example in Figure 7.24, most of the data seems to have originated from the north east of South Africa, with hubs along the east coast and in the Western Cape, and isolated data points in the centre of the country. As was to be expected with 75.1% of the documents bearing negative sentiment, the dominant colour on the map is red. The prevalent negative and smaller positive and neutral regions seem to be relatively evenly spread across the spectrum, with little evidence of any particular hotspot. The name of each location (that was identified as *Branch Name* by the user on the *upload and categorise* page of the ECCO system) is shown upon hovering the mouse over a certain geographical coordinate, along with the number of *positive*, *negative* or *neutral* documents associated with this location, depending on the circle over which the mouse is hovered. In the figure, the user is currently hovering the mouse over the *Springbok* location, which is associated with six documents of the *negative* sentiment class.

The map for the graph was generated using `Mapbox` [187], an open source mapping platform for customised maps. Country borders, roads and the names of various cities, regions and mountain ranges are automatically plotted on this map. Furthermore, the user is able to pan, zoom and tilt the map in order to obtain the desired observation angle. In Figure 7.25, for example, the map has been zoomed in and tilted to show the Cape Peninsula. Furthermore, the *negative* trace has been isolated and the location names are displayed as indicated in the legend in the top left-hand corner of the graph. Each geographical coordinate is now labelled with its allocated name followed by the total number of documents associated with this location. By utilising these features, the user is thus able to obtain a sufficiently detailed or broad view, according to the situational requirements.

Tabs 3 and 4 facilitate Processes 17.1 and 17.2 of the ECCO framework by identifying the data types of the variables to be plotted and then displaying graphs that exploit and visualise the characteristics of each data type. Finally, in the *multivariate analysis* tab of the *Dashboard* interface, the user is afforded the ability to identify possibly hidden relationships in the structured data component by analysing the results of a fitted statistical model (Process 18.0 of the ECCO framework). In this instantiation of the framework, a classification tree is used for this purpose (see §3.3.2).

As shown in Figure 7.26, the user is able to adjust the values of various hyperparameters of the decision tree. These include the criterion employed to split the data at each node in the tree, which can be set as the Gini index or information gain (the cross-entropy measure), as well as the splitting strategy. In particular, either the *best split* or the *random split* strategy may be selected. The former calculates the purity achieved by splitting a feature on every possible threshold (*e.g.* splitting the *age* feature at $1, 2, \ldots, 30, 31, \ldots$ years), whilst the latter only considers a number of randomly selected thresholds for each feature (*e.g.* splitting the *age* feature at $3, 10, 31, 47$ and 50 years). In either case, the best of the evaluated splits is selected. Furthermore, the *class weights* may be set to *balanced* or *unbalanced*. If a class imbalance prevails in the data, the decision tree is likely to predict only the majority class. By training the tree in respect of a subset of the data that exhibit an equal number of *positive*, *negative* and *neutral* observations (where the classes are *balanced*), this may be avoided. The maximum depth of the decision tree may also be set, along with the fraction or number of observations that are required to be on

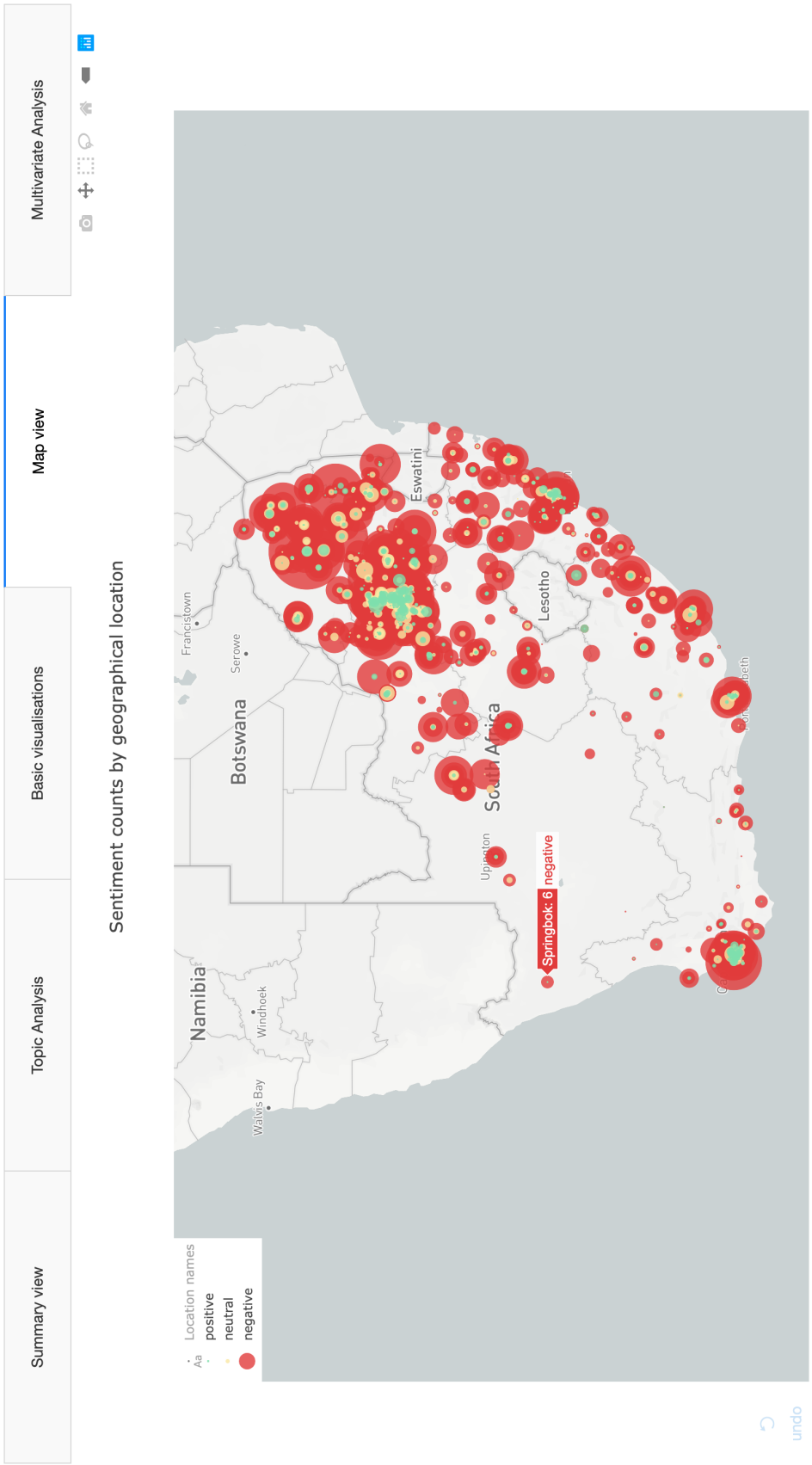


FIGURE 7.24: *The map view tab of the Dashboard interface. A bubble map illustrates the number of positive, negative and neutral reviews associated with each geographical location.*

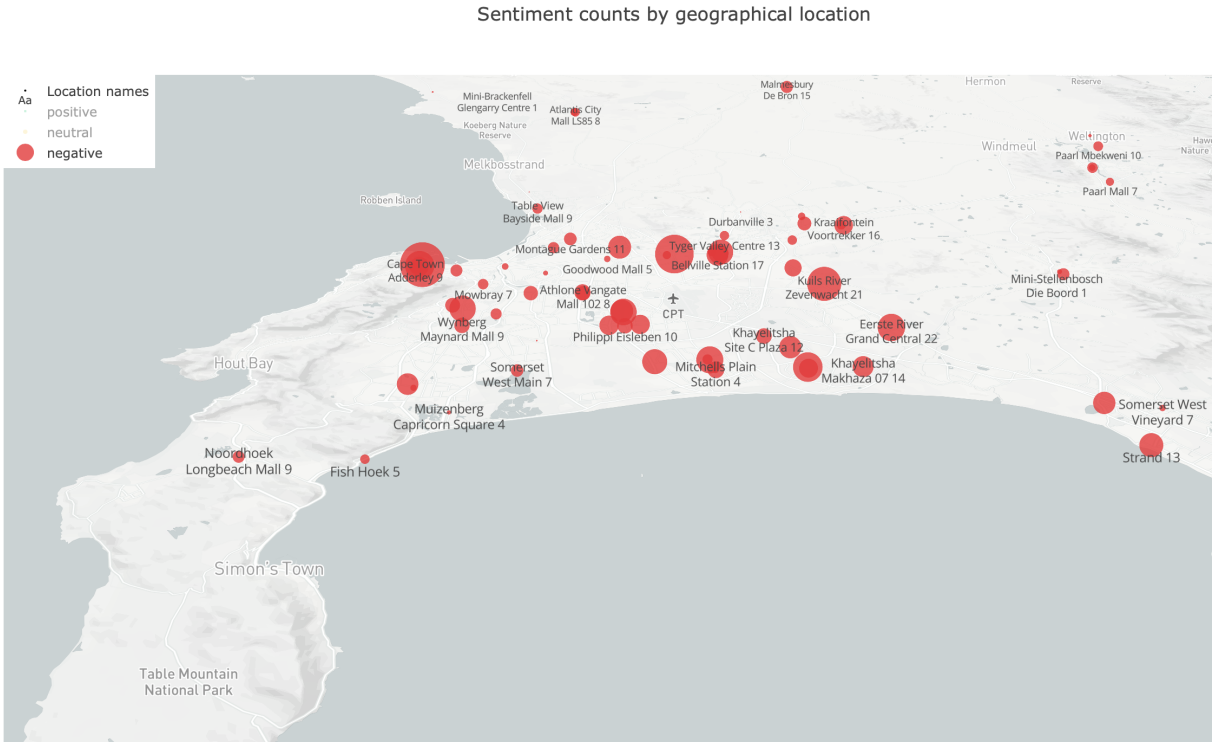


FIGURE 7.25: *A zoomed-in view of the map in Figure 7.24. The negative sentiment class has been isolated and the location names have been superimposed on the map.*

either side of a split region or in the leaf node of the tree. Finally, the maximum fraction or number of features to be considered during each split may be adjusted. If this number is set to 0.8, for example, as it is in the figure, a randomly selected subset of features (amounting to 80% of the total features) is considered during each split. Reducing this number can lead to a decrease in computation time and curb overfitting, but may also inhibit the performance of the classifier.

Upon clicking the *display tree* button, a decision tree is fit to the structured component of the data set with the sentiment class as the target variable. The decision tree classifier from the `Scikit-learn` library was used to implement this model, which is trained by means of an optimised version of the CART algorithm. Unlike the original CART algorithm, however, this implementation does not support categorical variables. Qualitative variables are therefore first transformed to a one-hot encoded representation. The variable *gender* for example, is transformed into two binary variables *gender_male* and *gender_female*. In order to reduce the correlation between columns[16], the first of such a group of transformed variables is excluded from the analysis. The decision tree is then fit on 80% of the data and tested in respect of its classification accuracy on the remaining 20% of the data. The accuracy of the model is relayed to the user, along with a representation of the resulting model in the form of a tree diagram, as shown in Figure 7.26. The user is also provided with a guide on how to interpret the categorical variables in this representation. More specifically, the notation $X\_Y < 0.5$ for a categorical variable X and its corresponding value Y means that the encoded binary variable X_Y is equal to zero. This, in turn means that the value of variable X is not equal to Y. The *False* branch emanating from such a node therefore represents the case where $X = Y$, whilst the *True* branch represents the case wehre $X \neq Y$.

[16]For a qualitative variable with n possible values, knowing the value of $n-1$ of the resulting one-hot encoded variables would enable a perfect prediction of the remaining variable.

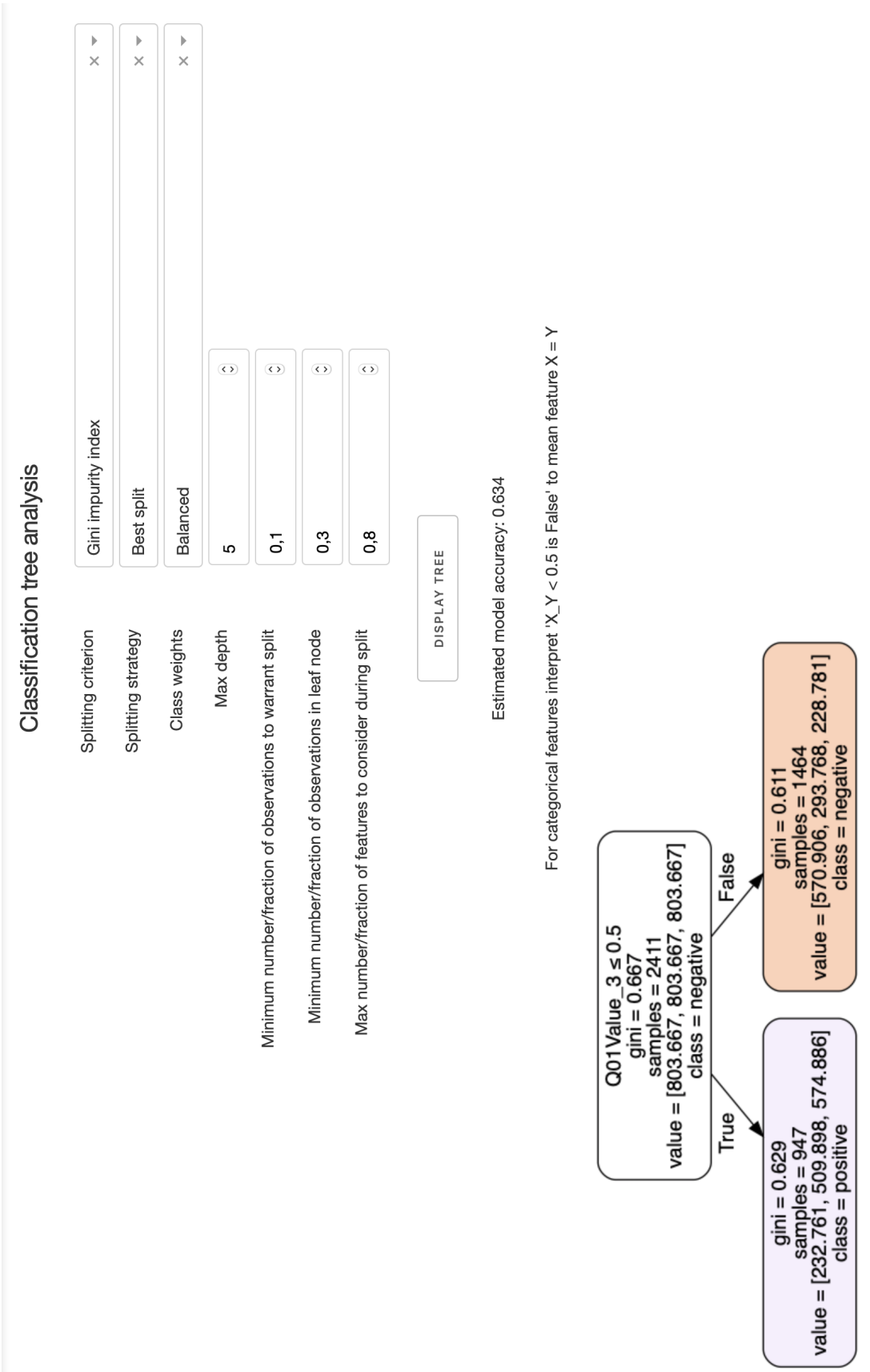


FIGURE 7.26: *The multivariate analysis tab of the Dashboard interface. The hyperparameters may be adjusted using the input fields to tune the decision tree model visualised below.*

At each node, the variable or feature which is split at the node is given, along with the threshold value at which it is split. Furthermore, the measure of purity achieved by the split is shown, along with the total number of samples in this branch, as well as a breakdown of the number of samples in each sentiment class (in the order *negative*, *neutral*, *positive*). Finally, the predicted sentiment class of all of the observations that fall within this branch is given. The colour of each node represents the predicted sentiment class of that node and the opacity of the colour represents the level of confidence the model has in this classification.

In the example in the figure, the decision tree achieved a test accuracy of 0.634. Whilst this can be used as a guideline of the measure of confidence of the model, the counts in each sentiment class for a branch can be used to gauge the confidence in a particular split (as represented by the opacity of the node). The top split is on the feature *Q01Value_03*. The model seems relatively certain that observations for which the variable *Q01Value* is equal to 3, are *negative*. It is much less certain of its prediction that observations for which this variable is not equal to 3 are *positive* (since the number of positive and neutral observations in this node are relatively similar). By varying the hyperparameters of the tree, the user can begin to identify other patterns in the data related to the structured variables.

Each of the tabs of the *Dashboard* interface contain functions that may be used to extract information and actionable insight from the unstructured and structured components of the input data. Even if only the unstructured data (the documents) are available for analysis, the *summary view* and *topic analysis* tabs can be used to promote an understanding of the sentiment expressed in the corpus. In order to gain maximum value from the analysis phase, however, the results of the individual tabs should be used to guide the analysis in other tabs. Topics identified by means of the *LDAvis* visualisation of the LDA topic model may, for example, be explored further *via* the topic-sentiment graph shown in Figure 7.21 or in any of the other tabs by filtering the reviews according to the keywords associated with that topic.

7.3 System verification and validation

In order to ensure the quality of the ECCO system, the approach outlined in §5.4 was followed. More specifically, the system was designed in a modular fashion according to the class diagram in Figure 7.1. Documentation of the system was, furthermore, executed by means of documentation strings within the code of the system, as well as the diagrams and descriptions in this chapter. Finally, the system was tested and validated according to the guidelines in §5.4.1 and §5.4.2.

Each of the classes was developed independently and the functioning of their methods tested using manually constructed test data (*program testing with test data*). Subsequently, the integration of these classes with each of the interface classes was tested using the same data (*link testing with test data*). This included the verification of the correct communication between the *Reviews* class and *SupplementaryData* class with both the *MainWindow* and *Dashboard* interfaces, as well as the correct functioning of launching the *Tensorboard* and *Dashboard* interfaces from the *MainWindow* interface, and the *LDAvis* interface from the *Dashboard* interface. Subsequently, the entire system was tested using these test data, before the case study of the following chapter was performed in respect of real-life data (*full systems testing with test/live data*). During this process, any calculations and data manipulations performed by the system were corroborated by manual calculations or other software such as `MS Excel`.

The best performing models developed during the case study by means of the ECCO system achieved test performances of over 85% accuracy and over 0.9 for the AUC value. Furthermore, the standard deviation was less than 2.3% for the accuracy and less than 0.021 for the

AUC score for all of the tested algorithms for 10 repetitions. These results were deemed very favourable compared with the performance achieved on other sentiment analysis tasks in the literature, which served as a form of validation. Furthermore, model performance in respect of three additional benchmark data sets was found to be competitive with state-of-the-art results published in the literature, as is described in the validation of the ECCO framework later in this dissertation.

As a further means of validation, the ECCO system and its case study results were presented to a subject matter expert in statistics, Professor Martin Kidd [156]. Using only the training data (for which true labels were available), he was able to reproduce some of the case study results returned during the analysis phase of the ECCO system using the statistical software `Statistica`. Furthermore, he confirmed the insights drawn from these results in a personal consultation.

Finally, the ECCO system was demonstrated to members of the data science department[17] of the industry partner affiliated with the case study, who found the system to be a useful tool for building sentiment analysis models and for analysing the results of these models.

7.4 Chapter summary

In this chapter, a proof of concept implementation of the ECCO framework, the ECCO system, was demonstrated. The technical implementation of the system was first described, including its object-oriented design and the libraries and frameworks that were used to develop it. The ECCO system was subsequently demonstrated in respect of the preprocessing and sentiment modelling stages that are facilitated primarily by the *MainWindow* interface, as well as the analysis stage that is primarily facilitated by the *Dashboard* interface. The implementation of each of these stages, including the models and parameters selected to populate the ECCO framework, was described in detail and demonstrated by means of screen shots of the actual system. The chapter closed with a brief outline of how the quality of the system was assured (verification and validation).

[17] Due to the confidentiality agreement signed with the industry partner, no reference is provided in this case.

Part III

Case study

CHAPTER 8

Case study background

Contents

In order to further illustrate the utility of the ECCO framework presented in Chapter 6, the instantiation of the framework, the ECCO system presented in Chapter 7, is applied to a real-world case study. In this chapter, a background of the case study is given, as well as a detailed description of the data associated with the study. Furthermore, the objectives pursued during the execution of the study are outlined. The results of the case study analysis are presented in the following chapter.

8.1 Background

As mentioned in Chapter 1, the communication model between an organisation and its customers is continually changing, rendering individual feedback from customers highly influential for overall customer satisfaction and company reputation. An increase in the influence of customer advocacy has also been observed in the financial industry. When seeking advice on banking products, for example, 71% of customers worldwide consult personal peer groups such as family and friends. Online communities are also an increasingly popular source of information, with just over half the clientèle of the financial sector making use of these resources globally [81]. The need for automated CRM systems is therefore also relevant in this sector.

The industry partner associated with this study is a South African retail bank with over 6 million customers[1]. In an attempt to monitor customer satisfaction levels, the bank launched an initiative to elicit customer feedback *via* SMS in association with a third-party vendor specialising in collecting and analysing customer experience (*voice of the customer*) data. The process by which the communication with customers is carried out is shown diagrammatically in the form of an *activity diagram* in Figure 8.1. In this diagram, rounded rectangles represent *activities*,

[1]The anonymity of the industry partner is protected by a non-disclosure agreement.

arrows represent *events* and diamonds represent *decisions*. The solid black circle and the black circle with a white ring represent the beginning and end of the process, respectively. Activities conducted by different entities (*i.e.* the customer, the bank and the third-party vendor) are separated into *swim lanes* by the dashed lines in the figure.

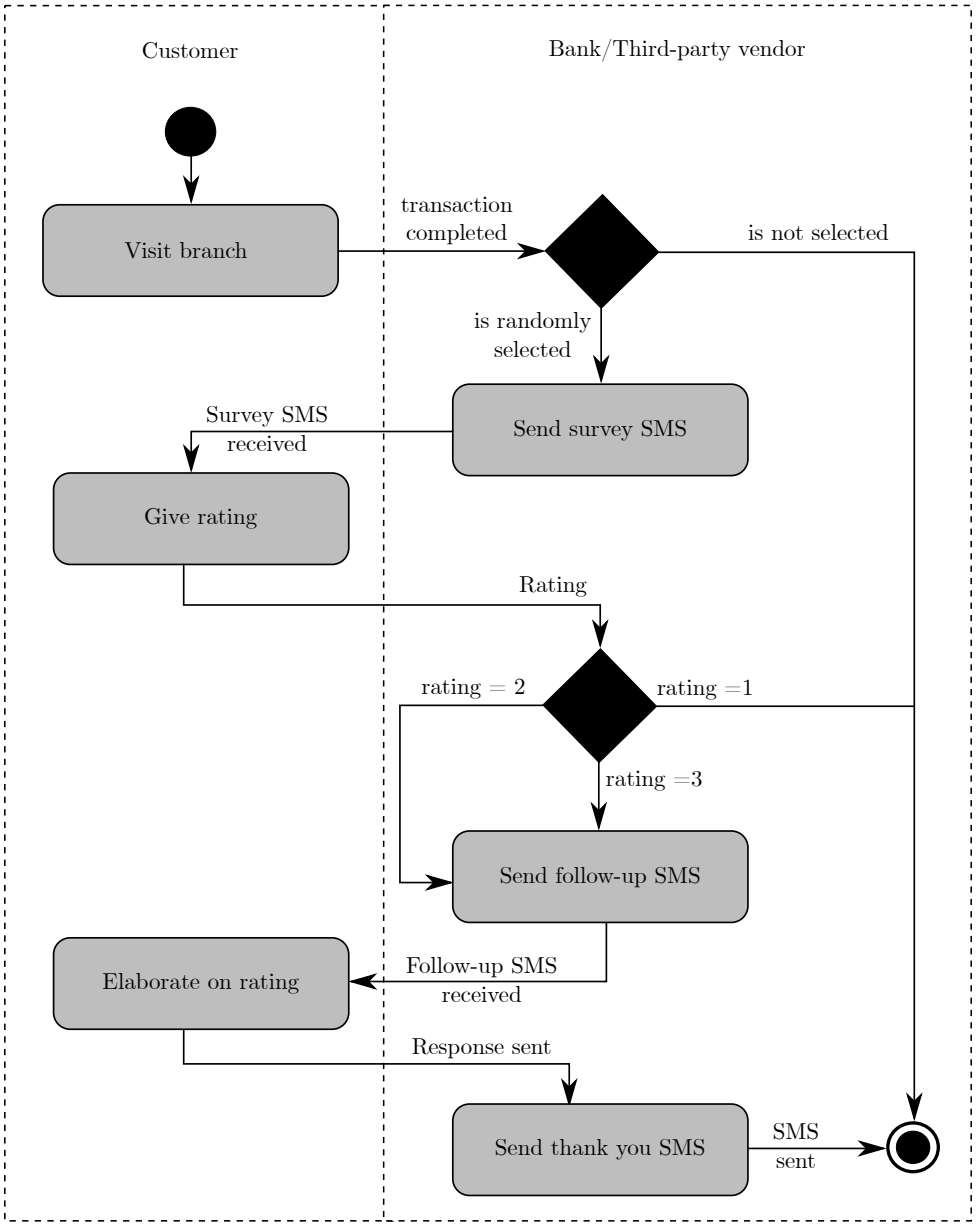


FIGURE 8.1: *An illustration of the process followed to collect the case study data from customers in the form of an activity diagram.*

After visiting a branch, randomly selected customers are sent an SMS requesting them to rate their experiences on a scale of 1 (*great*) to 3 (*bad*). Customers who respond with a negative rating (2 or 3) are then sent a follow-up SMS asking them to elaborate on their negative experiences and provide reasons for their rating (if the rating was 3), or suggestions as to how the bank may improve its service (if the given rating was 2). There is no follow-up on positive ratings in an effort to reduce costs.

The third-party vendor subsequently analyses the free-form customer responses to the follow-up messages using proprietary sentiment analysis software in order to assign sentiment scores of -1

(*negative*), 0 (*neutral*) or +1 (*positive*) to these responses. After analysing a batch of messages, however, it turned out that several customers had misunderstood the rating scale, assuming that a higher number translated to a positive rating, rather than a negative one. Therefore, although there had only been follow-up in respect of neutral or negative ratings, not all sentiment subsequently expressed in the messages was necessarily negative.

The bank is presently experiencing two primary problems with its current approach to customer satisfaction monitoring. First, upon investigation by the data science department of the bank, the third-party software used to analyse the sentiment expressed in the customer feedback messages was not deemed sufficiently accurate in classifying sentiment polarities. Secondly, after messages with negative sentiment are identified, employees have to examine these messages manually in order to gain actionable insight. Applying the ECCO framework may alleviate both of these problems by aiding the data science department in building and testing more accurate models for classifying sentiment tailored to the specific context, as well as by facilitating an in-depth analysis of the content of the messages in each sentiment category and the relationship between additional customer data and customer satisfaction without the need for manually reading each review.

8.2 Data preparation

The data employed in this case study were received in various separate data sets, which are described in Table 10.6. The data set *VOC Cleaned* contains details on the survey conducted in the manner described in Figure 8.1. More specifically, it contains attributes describing the customer's branch visit after which (s)he was contacted, including the name and type of the branch, the town and province in which the branch is located, as well as the customer's primary need for entering the branch and the type of consultant that was assigned to the customer. Furthermore, the data set contains the customer's initial response to the SMS (*i.e.* the rating of 1, 2 or 3), as well as the free-form text response (*Comment*) given as a result of the follow-up SMS, if applicable. A unique identifier for the customer (*ID*) is also included. Finally, the data set contains a *Sentiment* attribute. This is the sentiment score (−1, 0, or +1) assigned to the free-form text response by the third-party software.

The data set *SMS*, on the other hand, contains only those survey responses in respect of which free-form text responses were received, amounting to only 3.7% of the original 277 713 SMSs that were sent out. The attributes retained in this data set are the customer *ID*, the free-form response (*Comment*) and the software's sentiment score (*Numerical*). Furthermore, this score was converted to a description of the sentiment category (*Rating*) as either *positive*, *negative* or *neutral*. The data set *VOC labelled* contains a selection of 5 250 of the free-form text responses along with the associated customer ID (*CIF*) and the description of the rating assigned by the software (*Software_Sentiment*). Furthermore, this data set contains 500 labels that were assigned by a human annotator from within the bank's data science department in order to evaluate the software's rating.

The *Branches* data set constitutes a mapping of unique identifiers for the bank's branches to their geographical coordinates. Finally, the data set *Client demographics* contains several descriptive attributes pertaining to the customers in the bank's database, including personal attributes such as age, gender and salary, as well as banking-related attributes, such as the client's current loan status and average monthly banking fee. The attribute *CIF_NUMBER* is the same customer ID used in the other data sets. The remaining customer attributes are described in the following case study analysis chapter where an understanding of these attributes is necessary.

Data set name	Description	Records	Attributes
VOC Cleaned	Details of survey	277 713	*CreatedDate* *Branch.UniqueId* *Branch Name* *Branch Province* *Branch Type* *Contact.CIF_NUMBER* *Primary Need* *Consultant Type* *Q01Value* *Comment* *Sentiment*
SMS	Customers' free-form responses to follow up SMSs	10 363	*ID* *Comment* *Rating* *Numerical*
VOC labelled	A collection of annotated customer responses	5 250	*CIF* *M_Sentiment* *Software_Sentiment*
Branches	Mapping of branch IDs to geographical coordinates	5 250	*Branch UniqueId* *Latitude* *Longitude*
Client demographics	Database describing the bank's clients	277 713	*CIF_NUMBER* *Dependents* *City* *Postal_City* *Client_Deposit_Status_Desc* *Handed_Over_Status* *Client_Loan_Status* *Last_Loan_Branch_Key* *Last_branch_visit* *Salary_Status_Desc* *Salary_Freq_Desc* *Salary* *Gender_Desc* *Marital_Status_Desc* *Age* *Country_Description* *Network* *Service_Plan* *Internet_banking_registration_status* *Internet_banking_usage_status* *Num_Clients* *Most_Frequent_Branch_Key* *Average_Monthly_Fee* *New_Client_Date* *Client_Bank_Name* *Most_Frequent_Branch_Province* *Most_Frequent_Branch_Town* *Most_Frequent_Branch_Type*

TABLE 8.1: *The data sets received from the industry partner for the case study analysis.*

8.2.1 Annotating customer responses

As was discovered by the data science department of the industry partner, the third-party software did not seem to yield sufficiently accurate sentiment polarity classifications of the messages sent in by customers. Compared with the 500 manually labelled entries in the *VOC labelled* data set, the human annotator and the software were in agreement in only 60.4% of the cases. In order to formally evaluate the accuracy achieved by the software, and in order to evaluate and compare the models developed using the ECCO system, a *ground truth* data set was required.

This is a particularly challenging task in the case of sentiment analysis since, in contrast to other classification tasks, human annotators typically agree on a sentiment label only 80% of the time [54, 148, 204]. An attempt was, however, made to reduce the uncertainty associated with the assigned labels. A group of annotators was gathered from within the Department of Industrial Engineering at Stellenbosch University, representing diverse cultural backgrounds, an age range of approximately 20 years and both genders. A total of 2 500 reviews were then randomly selected from the data to serve as the *ground truth* data set. Each review was labelled as *positive*, *negative* or *neutral* independently by at least two human annotators. If these annotators agreed on a sentiment class, this class was assigned as the *true* class label. If not, a third annotator was asked to classify the review. If this annotator agreed with one of the first two annotators' classifications, then that class was assigned as the true class label. If not, additional annotators were asked to assess the review until a majority vote was achieved for one of the classes.

After the first round of annotations (with two annotators per review), the agreement between annotators was 81.8%, which is consistent with the literature. Furthermore, 11.8% of the labels assigned to reviews (two per review) were *positive*, 69.5% were *negative*, 15.1% were *neutral* and 3.6% were marked as *uncertain*. After the second round of annotation (where additional annotators were added in cases of disagreement), a label was found for each of the 2500 reviews. A total of 283 (11.3%) of the reviews were classified as *positive*, 1 819 (72.8%) were classified as *negative* and 398 (15.9%) were classified as *neutral*.

8.2.2 Merging the available data

From Table 10.6 it is clear that the data received from the industry partner were not presented in a normalised form (many of the data attributes were repeated across several data sets). Furthermore, only some of the data are relevant for the case study.

In order to use the data as input to the ECCO system, they needed to be transformed into two relational data tables, namely *review data* and *supplementary data*. Upon examining the data, four independent data sets were identified that are related to each other in the manner illustrated in the *entity relationship diagram* in Figure 8.2. The diagram may be interpreted as follows. Clients and branches each represent *primary* (stand-alone) entities, indicated by the bold outline of these entities in the diagram. Survey responses, on the other hand, contain attributes that describe the clients' responses to the survey, such as *CreatedDate* and *Q01Value*. Furthermore, each entry in the survey response table makes reference to a particular branch and customer associated with the entry. A branch or customer may be associated with several survey responses, as indicated by the *crow's foot* at the end of the connector between these entities and

the survey responses entity. Finally, each survey response *may*[2] be associated with an entry in the SMS data table, which contains clients' free-form text responses to follow-up messages (the unstructured data to be used in the case study).

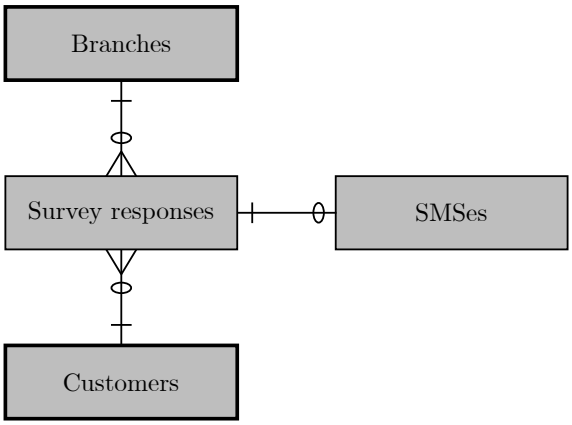


FIGURE 8.2: *An entity relationship diagram of the case study data.*

In order to create the two tables required for the case study, the clients data set was chosen as the *supplementary data* set[3]. The remaining entities were merged to form the *reviews* data set, which contains the attributes listed in Table 8.2. These constitute all the attributes contained in the *VOC cleaned* data set, but only for those records that contain a *Comment* field. A new variable $CreatedDay \in \{Monday, Tuesday, \ldots, Sunday\}$ was extracted from the *CreatedDate* variable, as it may be interesting to investigate whether weekly patterns can be observed in the data. Furthermore, the value of the *Sentiment* attribute was replaced with the labels assigned to the selected comments during the data labelling process described in §8.2.1. Finally, the latitude and longitude values for each of the branches were added to this data set.

Data set name	Description	Records	Attributes
Reviews	Data relating to the	10 354	*Contact CIF Number* *Comment* *Sentiment* *CreatedDate* *CreatedDay* *Primary Need* *Consultant Type* *Q01Value* *Branch UniqueId* *Branch Name* *Branch Province* *Branch Type* *Latitude* *Longitude*

TABLE 8.2: *The reviews data set created during data preparation for the case study.*

[2]If a zero is placed on the connecting line between two entities, it indicates that a *one-to-zero* relationship may also exist between these entities [147]. In other words, not every entity on the opposite end of the zero is necessary linked to the entity on the other end.

[3]Note that the *branches* data set also qualifies for selection as the supplementary data set. Due to the limited number of attributes in this data set, however, it was merged with the review data instead. In future implementations of the ECCO framework, the option of employing several supplementary data sets will be considered.

Those records in the reviews data set that did not contain valid entries for all attributes (with exception of the *Sentiment* attribute) were deleted. Consequently, nine of the 10 636 free-form text responses from the *SMS* data set were excluded from the analysis. Unfortunately, this included two of the responses that were labelled by the team of annotators, resulting in only 2 498 (24%) of the responses being labelled.

8.3 Objectives of the study

The strong bias towards the negative sentiment class exhibited by the data was to be expected, considering the fact that only customers who had responded with a non-positive rating were asked to submit another response during the data collection process. Positive comments therefore most likely came from customers who misunderstood the original rating scale. Any results returned by the ECCO system regarding the data must be interpreted within this particular context.

An analysis of the *VOC_cleaned* data set revealed that 84% of customers who responded to the initial survey question had rated their experience at the bank with a 1 (*great*). A further 11% responded with a neutral rating and only 5% of the 277 713 respondents gave a negative rating. Moreover, of the 44 309 customers that were asked to send a more detailed rating, only 10 363 (23%) actually did so. The distribution of sentiment in the *Reviews* data set analysed by means of the ECCO system is therefore not to be viewed as a representation of the overall customer satisfaction levels in this particular case, but may rather be used to identify the *true negatives* within the responses examined and analyse possible reasons for this negative sentiment.

The motivation for this case study is twofold. First, the sentiment model provided by the third-party vendor was to be evaluated against the models developed within the ECCO system. Secondly, an analysis of the customer reviews received from the industry partner was to be conducted using the results of one of these models in order to gain insight into the reasons why customers are dissatisfied with their experience at the bank. In pursuit of these objectives, the following questions had to be addressed:

(i) Is it possible to develop a model that outperforms the sentiment analysis software of the third-party vendor using the ECCO system?

(ii) Which types of models perform better in this context?

(iii) How many of the reviews received from customers did, in fact, contain negative comments?

(iv) What are the reasons for the dissatisfaction of customers in these cases?

(v) Are there any other distinguishable trends indicating that certain branches or customer profiles are more dissatisfied than others?

8.4 Chapter summary

In this chapter, a real-world case study to be conducted in the next chapter in order to demonstrate the value of the ECCO framework was introduced. The context of the study, namely feedback from customers of a retail bank, was first presented, detailing the manner in which the data were collected and describing the available data. Subsequently, the process followed to prepare the data for analysis was described, including the generation of a *ground truth* annotated

subset of the data and the merging of various data sources. Finally, the objectives that are to be pursued during the case study analysis were outlined.

CHAPTER 9

Case study analysis

Contents

In this chapter, the analysis results pertaining to the case study introduced in Chapter 8 are presented. During this process, the utility of the ECCO system (and thus also of the ECCO framework) is showcased, highlighting possible applications of the system. The processing steps performed on the data are first described, and this is followed by an account of the model development process. Finally, the results returned by the selected model are analysed, guided by the case study objectives outlined in §8.3. The chapter closes with a summary of the most important results of the case study.

9.1 Processing the data

As is shown in Figure 9.1, the *Reviews* data set from Table 8.2 was uploaded as the *review data* in the ECCO system, whilst the *Client demographics* data set from Table 10.6 was uploaded as the *supplementary data*. With respect to the *review data*, the *Comment* and *Label* fields were identified as the documents and their sentiment class labels, respectively, whilst the *Latitude*, *Longitude* and *Branch Name* variables were selected to describe the geographical coordinates and the associated location name, respectively. Furthermore, the *CreatedDate* attribute was categorised as a date variable. The remaining variables in the *Reviews* data set were classified as qualitative variables.

With respect to the *supplementary data*, the variables *Age*, *Salary* and *Average_Monthly_Fee* were identified as quantitative variables, whereas the following variables were selected as qualitative variables:

(i) *City*: The client's city of residence,

(ii) *Country_Desc*: The client's country of residence,

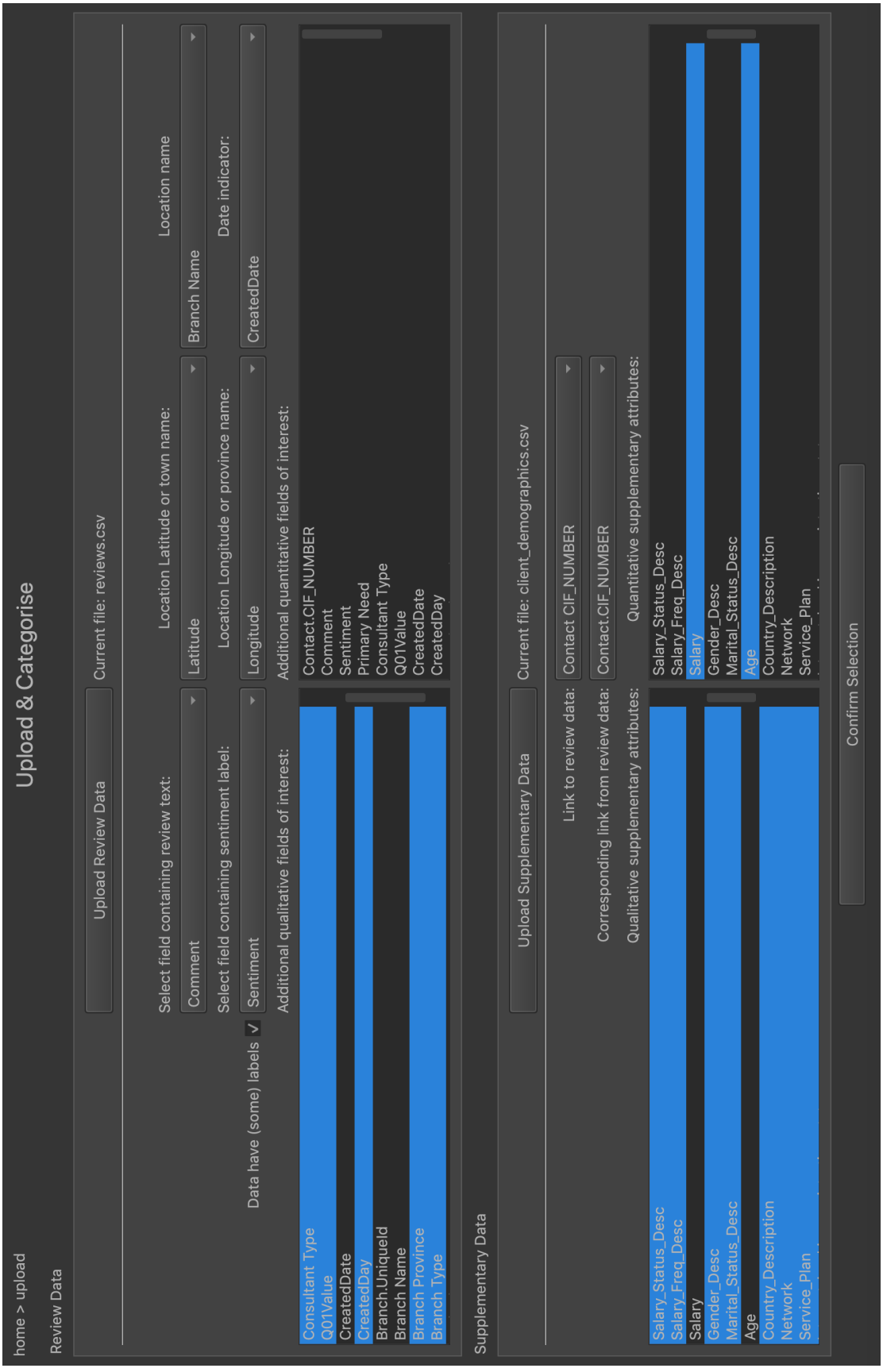


FIGURE 9.1: *The categorisation of the case study data.*

(iii) *Gender_Desc*: The client's gender,

(iv) *Marital_Status_Desc*: The client's current marital status,

(v) *Salary_Status_Desc*: How the client receives his or her salary (*e.g.* whether it is paid into his or her bank account),

(vi) *Salary_Freq_Desc*: Whether the client is paid weekly, fortnightly or monthly,

(vii) *Client_Bank_Name*: The name of the client's primary bank,

(viii) *Network*: The client's cellphone service provider,

(ix) *Client_Deposit_Status_Desc*: The status of the client's account (*i.e.* whether the client's account is active, dormant *etc.*),

(x) *Handed_Over_Status*: Whether the loan account has been handed over to debt collectors in the case of non-payment (or to what extent such a handover is in progress),

(xi) *Client_Loan_Status*: Whether the client currently has an active loan and whether payments are in arrears,

(xii) *Service_Plan*: The client's account type,

(xiii) *Internet_banking_registration_status*: Whether the client has registered for internet banking,

(xiv) *Internet_banking_usage_status*: Whether the client makes use of internet banking,

(xv) *Most_Frequent_Branch_Province*: The province in which client's most frequently visited branch is situated,

(xvi) *Most_Frequent_Branch_Town*: The town in which client's most frequently visited branch is situated, and

(xvii) *Most_Frequent_Branch_Type*: The type of the client's most frequently visited branch.

Some variables were excluded from the study due to redundancy (*e.g. Postal City* was deemed similar to *City* and the other, more interpretable descriptors of a client's most frequent branch were favoured over *Most_Frequent_Branch_Key*, which only provides a mapping of the branch to these other attributes), due to lack of value for the analysis (*e.g. Num_Clients* had a value of 1 for all clients in the data set), or due to their data type (*e.g.* the date variables *New_Client_Date* and *Last_branch_visit* are not well suited for the count plots or box plots employed in the ECCO system). The primary key of the *Client demographics* data set *Contact CIF_NUMBER* was, finally, identified as such in both data sets in order to facilitate the merging process.

As is shown in Figure 9.2, 15 364 (277 713 − 262 349) duplicate records were removed from the data set. Upon closer review of the original data set, these records constituted *null* values or blank entries. Furthermore, 5 740 (262 349 − 256 609) records were missing more than 50% of their entries. The cleaned data set therefore comprises 156 609 records and the twenty one attributes described previously.

The original corpus (the *Comment* column of the *reviews data*) contained 12 448 unique tokens (*types*) between the 10 354 documents. The word cloud of the unprocessed corpus is shown in Figure 9.3. From this figure, it is evident that stop words such as *and* and *it* are the most frequent words in the corpus. In order to determine the effect of various preprocessing steps

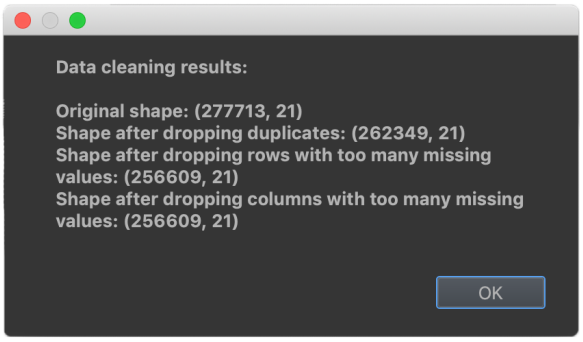


FIGURE 9.2: *The effects of the data cleaning process on the selected attributes of the Client demographics data set.*

on the number of unique tokens, only the tokenisation step was applied at first. Each of the other available preprocessing operations was then added to the selection iteratively, evaluating the effects of the sequence of operations in each case.

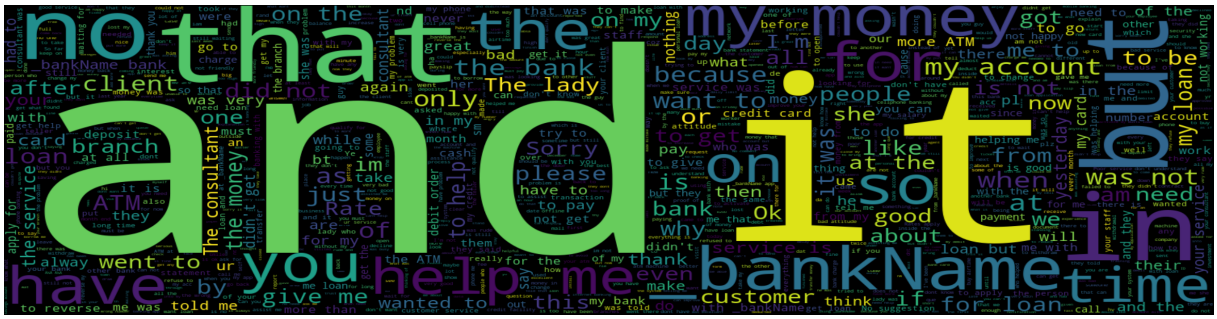


FIGURE 9.3: *A word cloud illustrating the most frequent tokens in the original, unprocessed corpus.*

A summary of these effects, expressed as the total number of types in the corpus, is given in Table 9.1. From the table, it is clear that 344 (12 448 − 12 104) unique tokens were removed from the corpus with the removal of the `NLTK` stop words. Although the list itself consists of only 175 words, the words are removed without regard for case (whilst the tokens "*It*" and "*it*" are considered to be two distinct types, both words would be removed from the corpus based on the stop word "*it*"). A further 26 (12 104 − 12 078) types were removed from the corpus with the removal of punctuation marks. Since the `NLTK` stop word list is not tailored to a sentiment analysis problem, however, certain words were excluded from this list. These include the words *but, no, nor, not, don't, aren't, couldn't, didn't, doesn't, hadn't, hasn't, haven't, isn't, wasn't, weren't, wouldn't,* and *won't*, which could indicate shifts in sentiment, as well as the terms *above* and *below*, which could also be used to indicate sentiment (*e.g. "the service was below standard"*). Finally, the intensifiers *too* and *very*, and the exclamation mark were also excluded from the list. With the amended list, 24 (12 102 − 12 078) of the types that were removed in preprocessing Sequence 3 in Table 9.1 were preserved in Sequence 4.

The grouping of tokens with primarily numerical characters into a single token further reduced the size of the vocabulary by 605 types to 11 497. Since customers were asked to rate the bank on a scale from 1–3 in the original feedback request, however, the numbers 1, 2 and 3 were excluded from this aggregation. Overall, the filtering of the corpus (Sequences 1–6) reduced the size of the resulting vocabulary by 7.6% ((12 448 − 11 500) − 12 448). Applying case normalisation had a much greater effect, reducing the number of types by 2 096 (11 500 − 9 404). Applying the spelling algorithm in Algorithm 7.1 further reduced this number by 4 019 (9 404 − 5 385),

Sequence number	Processing steps applied	Number of types
1	Tokenisation	12 448
2	1 & stop word removal	12 104
3	2 & punctuation removal	12 078
4	3 with an amended stop word list	12 102
5	4 & grouping of numbers	11 497
6	5 with certain numbers excluded	11 500
7	6 with case correction	9 404
8	7 with spelling correction	5 385
9	8 & Porter stemming	3 951
10	8 & Lancaster stemming	3 484
11	8 & Snowball stemming	3 920
12	8 & Lemmatisation with WordNet	4 948

TABLE 9.1: *The effects of various preprocessing steps on the size of the vocabulary.*

amounting to over 30% of the original vocabulary size. This is testament to the fact that the data were ridden with spelling errors.

Finally, each of the available stemming or lemmatisation algorithms was applied to the filtered, case-normalised and spell-corrected corpus in Sequences 9–12. The claim that the Lancaster stemming algorithm is the more *aggressive* stemming algorithm, followed by the Snowball stemming algorithm and Porter's stemming algorithm (see §7.2.1) seems to hold true for the data set in the case study with these algorithms affecting a final vocabulary size of 3 484, 3 920 and 3 951, respectively. This effect is significantly smaller for the lemmatisation algorithm, which reduces the vocabulary to a final size of 4 948. In spite of this reduced normalisation effect, Sequence 12 was chosen as the final preprocessing sequence for the data, in light of the fact that one of the objectives of the case study was to determine which types of models perform better in respect of this data set. Since the lexicon-based models embedded into the ECCO system make use of pre-compiled sentiment lexicons, it is necessary that the preprocessed documents comprise words which are contained in the English dictionary. The word cloud of the resulting, preprocessed corpus is shown in Figure 9.4.

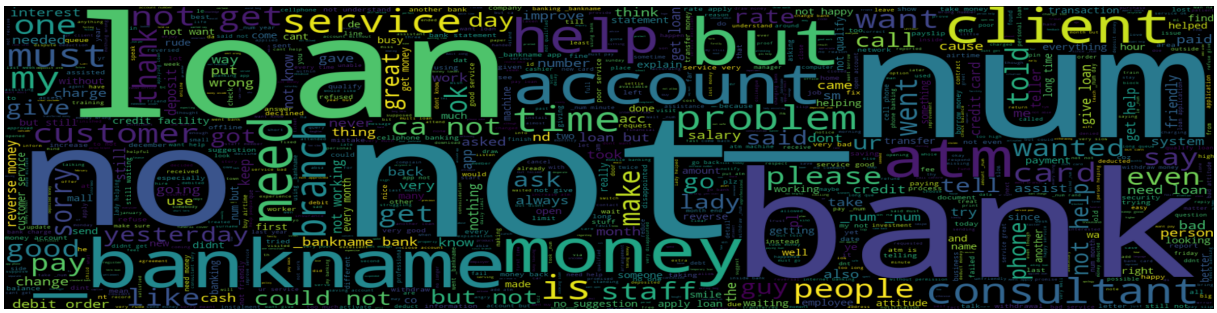


FIGURE 9.4: *A word cloud illustrating the most frequent tokens in the final, preprocessed corpus. The name of the industry partner was replaced by the token "_bankname" in order to preserve anonymity.*

Apart from the retained stop words *no*, *not* and *but*, and the aggregated token *_num*, words typically associated with a financial bank constitute the most frequently observed words in the corpus. These words include *bank, loan, money, account, service, branch, client* and the name of the bank representing the industry partner (which has been replaced by the token *_bankname* in the interest of anonymity). Terms which stand out more from the context are *help*, *problem* and

time, which may warrant further investigation during the analysis stage. Overall, the ECCO system was used during the preprocessing phase to reduce the vocabulary of the corpus by over 60% $((12\,488 - 4\,948)/12\,488)$ *via* filtering and normalisation, bringing information-bearing words into the forefront.

9.2 Developing a suitable sentiment classification model

One of the objectives of the modelling component of the ECCO framework is to facilitate an improved MST selection process, as described in §6.3.2. By enabling the user to test various model-hyperparameter-feature combinations rapidly during a single iteration, for example, the computational load on the user is reduced. Since the number of combinations that have to be evaluated by the grid search algorithm implemented in the ECCO system can quickly grow large, however, it is desirable to limit the number of values tested for certain hyperparameters or features during each iteration. In this section, the process followed in pursuit of this objective is first described. Subsequently, the results of the models generated by means of this limited grid search are presented and discussed with reference to the third-party software currently employed by the industry partner. Finally, various approaches to forming ensembles of these models are evaluated.

9.2.1 Improving the efficiency of the hyperparameter search

Empirically, two factors have a large influence on the computational time of the grid search. First, the metrics used to evaluate the performance of each fold incur a varying computational cost. Secondly, if the size of the vocabulary (the number of tokens or features used) grows significantly large, the time required to train the machine learning algorithms in respect of the term-document-matrix can become excessively long. The approach adopted during the case study therefore sought to limit the computational expense of each iteration by selecting the computationally most efficient metric and the smallest possible vocabulary size without significantly compromising on the performance achieved by the models.

For this purpose the smallest, computationally least expensive model, naïve Bayes, was evaluated for varying values of the vocabulary size. More specifically, nine experiments were performed for each value of the vocabulary size, evaluating three document models (based on term presence, term frequency and TF-IDF measures, respectively) and three n-gram ranges (unigrams, bigrams, and unigrams and bigrams). For each experiment, a grid search was performed (where applicable) during which the hyperparameter α was selected as one of the values in the set $\{0.0001, 0.2, 0.4, 0.6, 0.8, 1\}$ using 3-fold cross-validation. Each grid search was performed twice — once using accuracy (and thereby also the equivalent metrics recall, precision and F-measure) and once using the AUC score to evaluate performance in respect of each of the three folds. Whilst the AUC score is the preferred metric due to its invariance to class imbalance, the calculation of this metric is more expensive since it must be computed for each of the three classes separately and then averaged, as opposed to the accuracy metric, which is computed directly. As a measure of reference, performing one iteration of the grid search (fitting and evaluating three folds) according to the naïve Bayes algorithm for a vocabulary size of 50 took between 0.1 and 0.4 seconds using the accuracy metric, and between 0.4 and 0.9 seconds using the AUC metric, depending on the document model and n-gram range. For larger vocabulary sizes and more complex learning algorithms, which require several iterations of a gradient-based optimisation algorithm, this makes a significant difference.

The results of these experiments are illustrated in Figure 9.5, where each data point represents the highest AUC score achieved by the nine models for each value of the vocabulary size, both in the case where the grid search was performed to maximise AUC score and the case where the grid search was performed to maximise model accuracy. The training and test data were fixed for these experiments, since the objective was to estimate the *relative* effect of the vocabulary size on model performance.

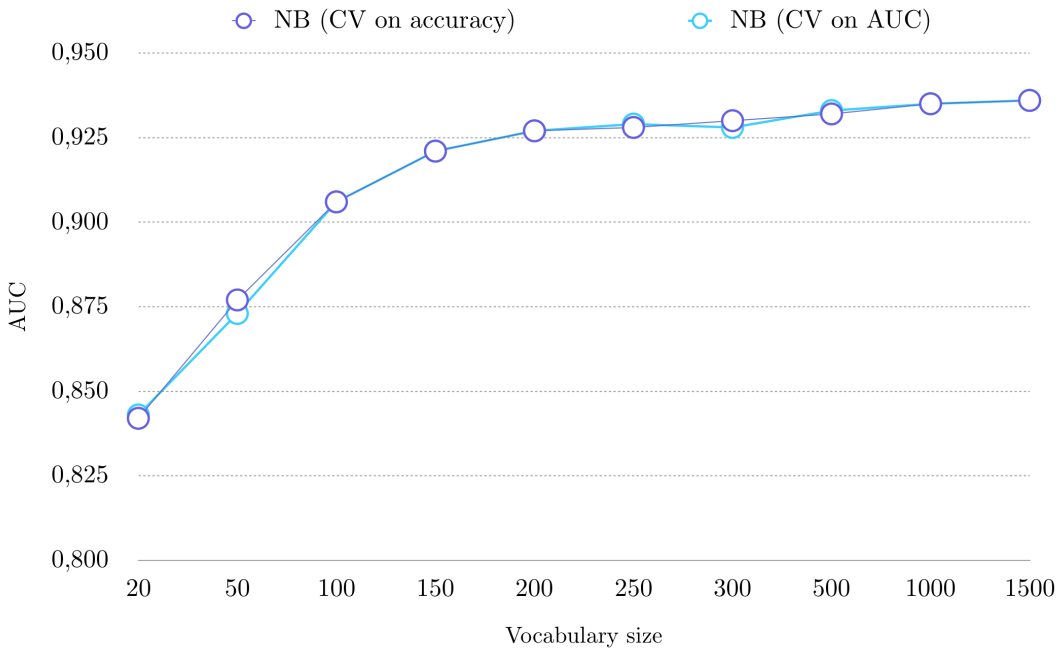


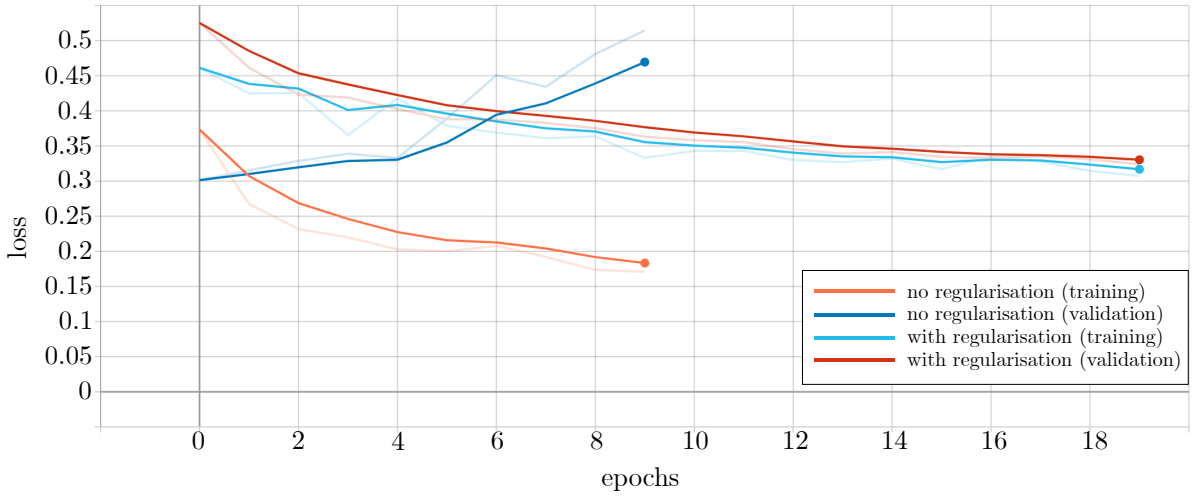
FIGURE 9.5: *The effect of vocabulary size on the AUC value achieved by a Naïve Bayes model in respect of a fixed test data set and a fixed training data set.*

As is evident from the figure, the AUC scores achieved by the models vary very little with the metric used during cross-validation. Furthermore, increasing the size of the vocabulary has a significant positive effect on performance between 20 and 200 tokens, but this effect reaches a plateau after a vocabulary size of approximately 200. Based on these results, the accuracy metric was employed for all subsequent grid searches. Moreover, a vocabulary size of 250 was provisionally selected for the case study analysis.

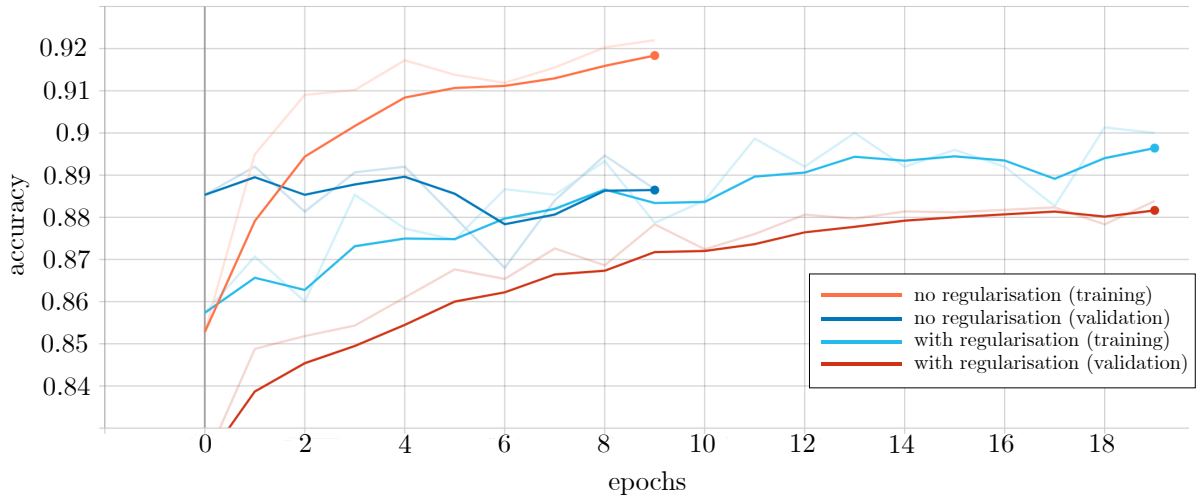
Due to the large number of hyperparameter combinations that could be formed for the three deep learning models (ANN, CNN and LSTM), the manual hyperparameter tuning function of the ECCO system was employed to identify good hyperparameter ranges prior to the grid search. During this process, the chosen vocabulary size of 250 was implemented, along with a simple unigram presence document representation for the ANN algorithm. A similar process was followed for all three algorithms. More specifically, a small, simple network was employed as a starting point. The graphs of the training and validation losses and accuracies were then scrutinised *via* the *Tensorboard* interface in order to inform the necessary changes that should be made to the model according to the guidelines for monitoring the learning process of neural networks outlined in §3.5.1. This process was repeated until a satisfactory outcome was observed (in terms of the loss and accuracy curves). In each case, the training data constituted 70% (1 750 observations) of the total labelled data and the validation data constituted 10% (250 observations) of the total labelled data. Having found suitable hyperparameter values for these

models, the appropriateness of the vocabulary size chosen based on the naïve Bayes classifier could be verified in respect of the other machine learning models.

The starting point for the ANN algorithm was a simple network with one hidden layer comprising ten neurons. The widely used ReLU activation function was applied to these neurons. Initially, no regularisation or batch normalisation procedures were applied. Furthermore, the popular ADAM optimisation algorithm was employed with an initial learning rate of 0.2, an initial learning rate decay of zero and an initial training duration of ten epochs. Examining Figure 9.6, it may be concluded that the network is indeed *learning* during training, since the training loss decreases and the training accuracy increases with the number of epochs. The validation loss, however, diverges as the number of training epochs increases. This indicates that the model may be overfitting the data. In an attempt to curb this effect, ℓ_2 regularisation was applied to the network with a small regularisation parameter $\lambda = 0.001$. Furthermore, since there was no evidence of a stagnation in the training accuracy after ten epochs, the number training epochs was increased to 20.



(a) *Loss*



(b) *Accuracy*

FIGURE 9.6: *Tensorboard graphs for ANN with and without regularisation. (a) The loss curves and (b) the accuracy curves of the first two tested network configurations for the ANN model generated by the TensorBoard interface are shown.*

These changes to the network structure had the desired effect on the graphs in Figure 9.6. The validation loss now followed the decreasing trend of the training loss and the validation accuracy followed the increasing trend of the training accuracy. The new loss curve, however, exhibited a shallower decrease, indicating (as described in §3.5.1) that the learning rate may be too large. The learning rate was therefore decreased from 0.2 to 0.02 during the next iteration. The results of these changes are shown in Figure 9.7.

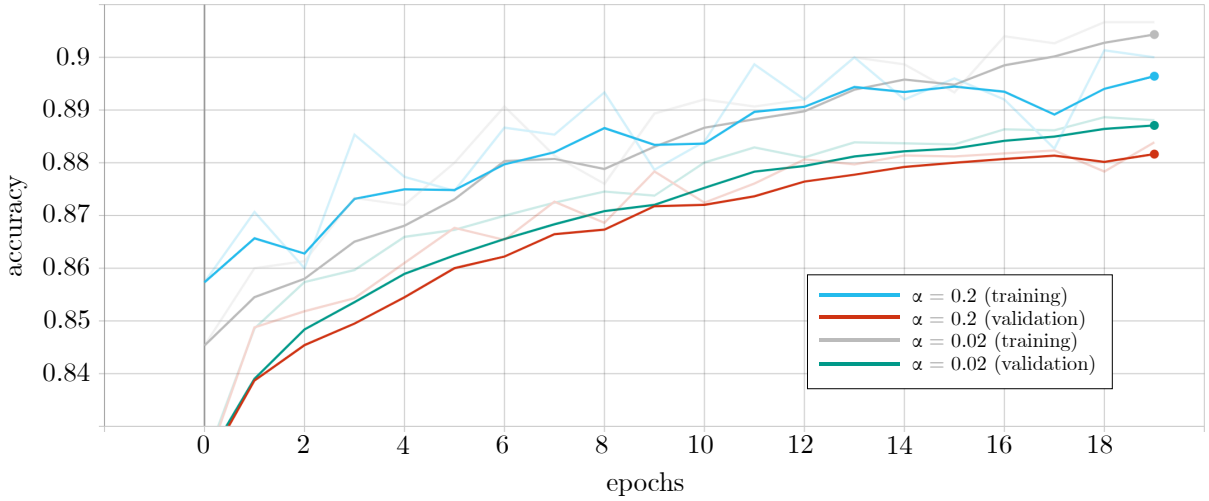


FIGURE 9.7: *Tensorboard accuracy curve comparing the performance of an ANN with a learning rate of 0.2 and an ANN with a learning rate of 0.02.*

The decreased learning rate resulted in both the training and validation accuracies increasing when compared with the previous network configuration, as shown in the figure. Given a working simple network, the complexity was then increased in order to determine whether the performance could further be improved by applying a larger network. A second hidden layer was therefore added, which also comprised ten neurons. It was anticipated that the model may take longer to train given the added complexity; therefore, the number of training epochs was increased from 20 to 30. As can be seen in Figure 9.8, the training and validation accuracies of this model were significantly higher than those of the previous model. Whilst both the training and validation accuracy increase until about the 10^{th} epoch, the network appears to start overfitting the data after this point, with the training and validation accuracies diverging[1]. Whereas parameter norm penalties were applied to curb overfitting of the initial network, another form of regularisation was employed in this case, namely early stopping. Training the network for only ten epochs results in a training and validation accuracy of over 90%. The number of epochs of the network were therefore varied around this number during the subsequent grid search. All other hyperparameter values from this network configuration were retained.

The effect of batch normalisation on network performance was also tested. Whilst this did improve the training performance, the validation performance was impacted negatively, exhibiting a *noisy*, fluctuating pattern throughout the training procedure. This is likely caused by the algorithm's inability to accurately estimate the values of the population mean and population variance during training, which are used to apply batch normalisation to the validation

[1]The fact that the validation accuracy is initially higher than the training accuracy can be attributed to the fact that the validation data were *well suited* for the network's configuration at the initial conditions. These validation procedures are not intended to yield an accurate score of the model's overall performance, but rather to determine whether learning takes place with respect to both seen and unseen data. Since the validation accuracy increases from its initial position in the graph, it may be concluded that generalisable learning has taken place during the first few epochs.

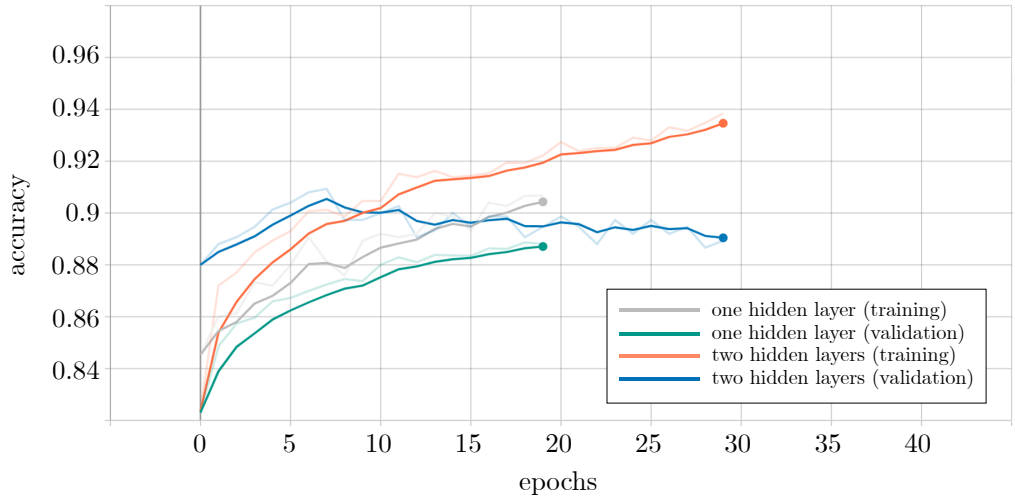


FIGURE 9.8: *Tensorboard accuracy curve comparing the performance of an ANN with one hidden layer comprising ten neurons and an ANN with two hidden layers each comprising ten neurons.*

data. During training, the sample mean and sample variance of the batch during each epoch is employed for batch normalisation and the population estimates are calculated by the moving average of these values as explained in §7.2.2. Altering the momentum value used in the calculation of these statistics did not result in any improvement. This seems to be a common problem [150, 149], especially in conjunction with the `Keras` library, as it was still an *open issue* [150] on the associated forums at the time of writing this dissertation.

A similar approach was followed for the CNN network, where the initial network architecture comprised an embedding layer of size 10 and one convolutional layer with ten filters, each performing a valid convolution with a kernel size and stride of 1. Initially, no pooling or regularisation was applied. The remaining hyperparameters (activation function, optimisation algorithm, learning rate (decay) and number of epochs) were set to the same values as for the initial ANN architecture. The resulting accuracy curves are shown in Figure 9.9. As is evident from the figure, both the training and validation curves are relatively flat, indicating that the learning rate is too low. After some experimentation, the learning rate was finally reduced from 0.2 to 0.01, which significantly improved training performance, as shown in the second set of graphs in Figure 9.9.

As is also evident from the accuracy curve and the loss curve of this network, shown in Figure 9.10(a), however, the model appears to be overfitting the training data with the validation loss increasing towards the end of the training period and the generalisation gap between the training and validation accuracies growing larger with the number of training epochs. In order to curb this effect, regularisation was applied in the form of ℓ_2 parameter norm penalties. A value of $\lambda = 0.001$ was first tested to no avail. Increasing λ to 0.01, however, achieved the desired regularisation effect with both the training and validation loss decreasing consistently throughout the training process. As expected, the training accuracy decreased slightly compared with that of the unregularised network, whilst the validation accuracy increased.

A pooling layer with a kernel size and stride of 2 was then added to the network in order to gauge its effect on performance. This, however, had a negative effect on both training and validation accuracies, as shown in Figure 9.11. A possible reason for this phenomenon is that dimensionality of the already small network is reduced through the application of valid convolutions — a further reduction by means of pooling likely caused too great a loss of information in the hidden layer. The number of filters was therefore increased from ten to twenty. Whilst this resulted in a

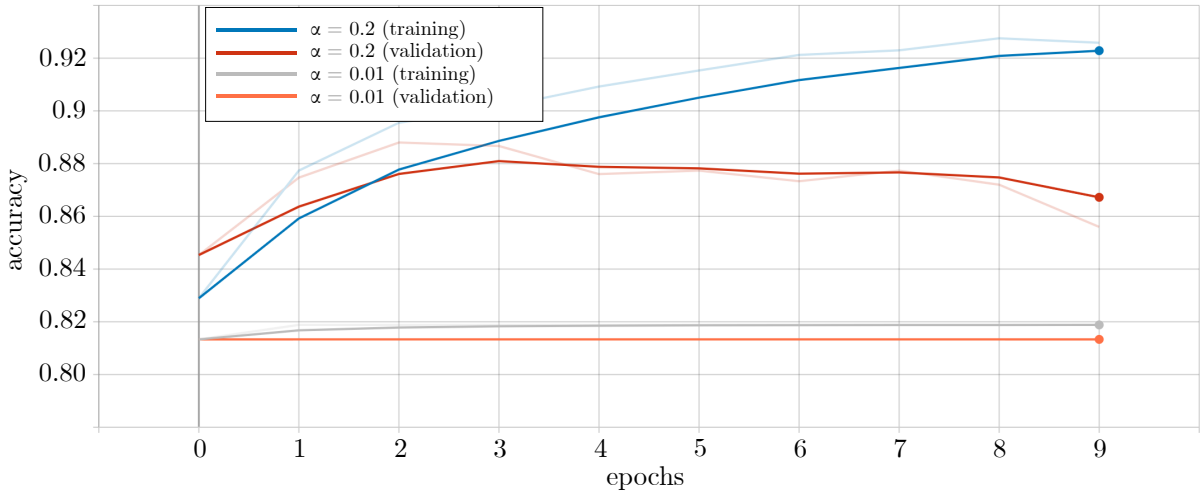


FIGURE 9.9: *Tensorboard accuracy curve comparing the performance of a CNN with a learning rate of 0.2 and a CNN with a learning rate of 0.01.*

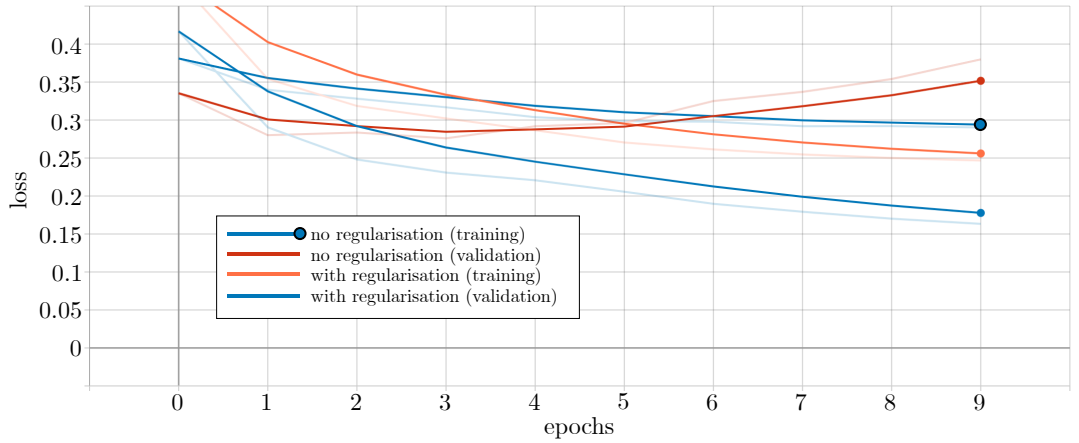
higher performance than in the network with ten filters and pooling, the network in which no pooling was applied still outperformed the other networks. Pooling was therefore not applied to the network.

Increasing the number of filters in the convolutional layer to twenty without applying pooling, however, improved performance slightly, as shown in Figure 9.12. Applying thirty filters, on the other hand, caused a decrease in both training and validation accuracies, as did an increase of the kernel size (regardless of the number of filters and the presence of a pooling layer). Finally, the application of batch normalisation had the same effect on the CNN as it did on the ANN. With a final training accuracy of 90% and a final validation accuracy of 89%, however, the performance of the network was deemed satisfactory.

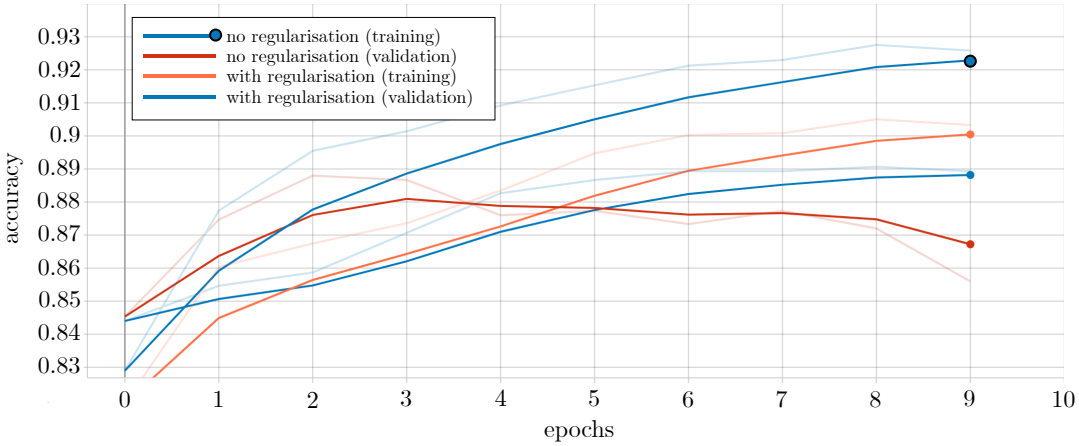
Finally, the initial network configuration for the LSTM model was an embedding layer of size 10 and an LSTM cell with one output unit, with hyperparameters related to the training set as in the case for the initial ANN and the initial CNN network configurations. As shown in Figure 9.13, the flat accuracy curves again indicate that the learning rate of 0.2 is too large. Decreasing the learning rate to 0.01 (after some experimentation) successfully alleviated this problem, as shown in the second set of graphs in the same figure.

The network seemed to exhibit some overfitting after the second epoch, however. In order to curb this effect, ℓ_2 parameter norm penalties were applied. Setting $\lambda = 0.01$ had little effect on the network. As shown in Figure 9.14, the validation loss curve exhibited a less pronounced divergence than the network without regularisation, but the final validation accuracy of the network did not improve. Setting $\lambda = 0.1$, on the other hand, caused both the training and validation accuracies to drop significantly, only showing evidence of improvement after the third epoch. Due to the unsatisfactory effects of parameter norm penalties, this regularisation technique was not implemented at this stage. Taking a different approach, the number of output layers was instead increased to ten.

As shown in Figure 9.15, this had a positive effect on performance, significantly increasing both the training and validation accuracies. With a training accuracy of almost 96% and a validation accuracy of close to 90%, however, the generalisation gap still proved to be large. After attempting to apply ℓ_1 and ℓ_2 regularisation, as well as dropout to no avail, it was decided that the regularisation method of early stopping would instead be applied by limiting the number



(a) *Loss*



(b) *Accuracy*

FIGURE 9.10: *Tensorboard graphs for CNN with and without regularisation. (a) The loss curves and (b) the accuracy curves of the second and third tested network configurations for the CNN model generated by the TensorBoard interface are shown.*

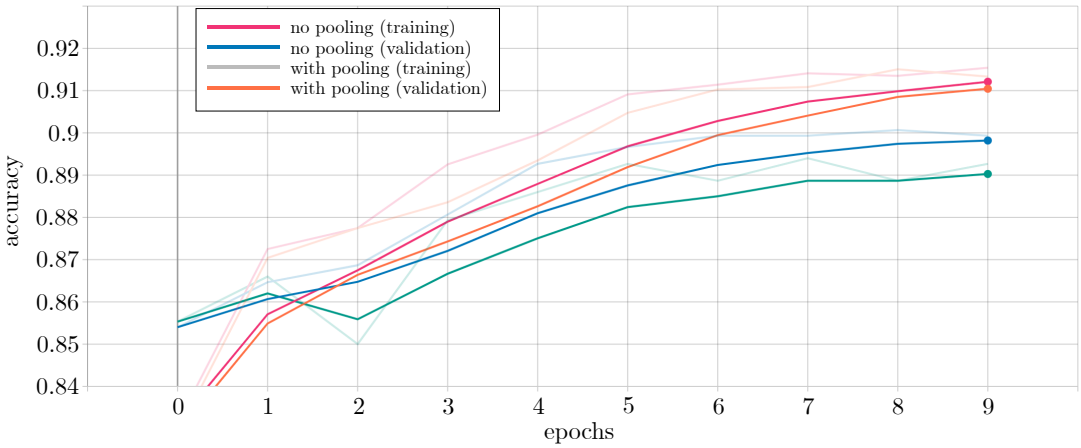


FIGURE 9.11: *Tensorboard accuracy curve comparing the performance of a CNN with and without the application of a pooling layer.*

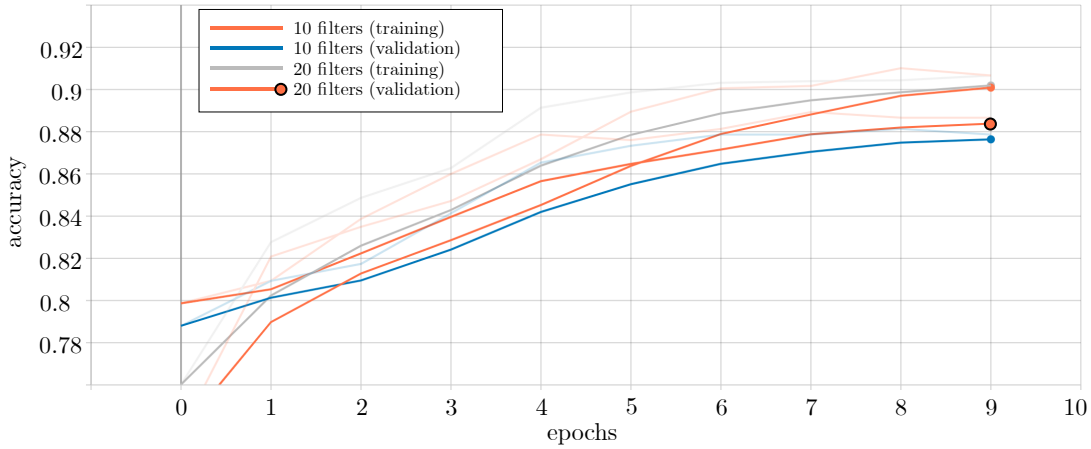


FIGURE 9.12: *Tensorboard accuracy curve comparing the performance of a CNN with ten filters and twenty filters in the convolutional layer, respectively.*

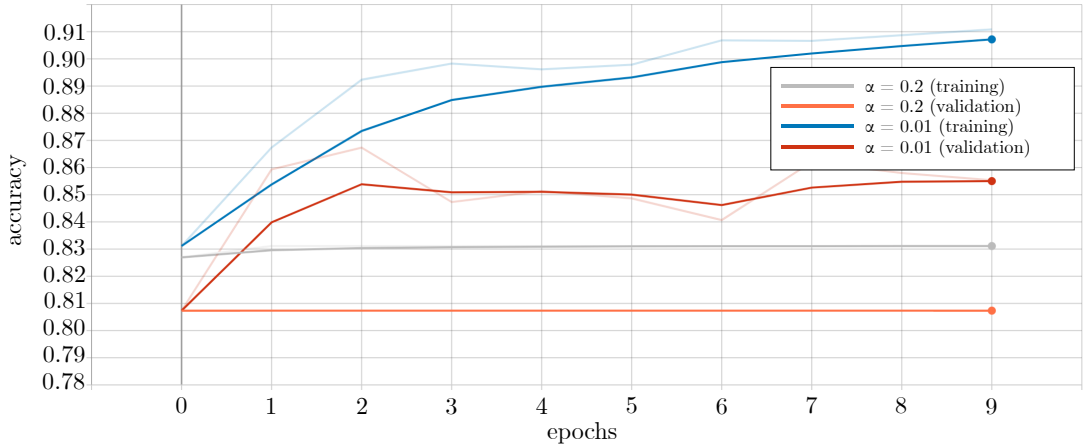
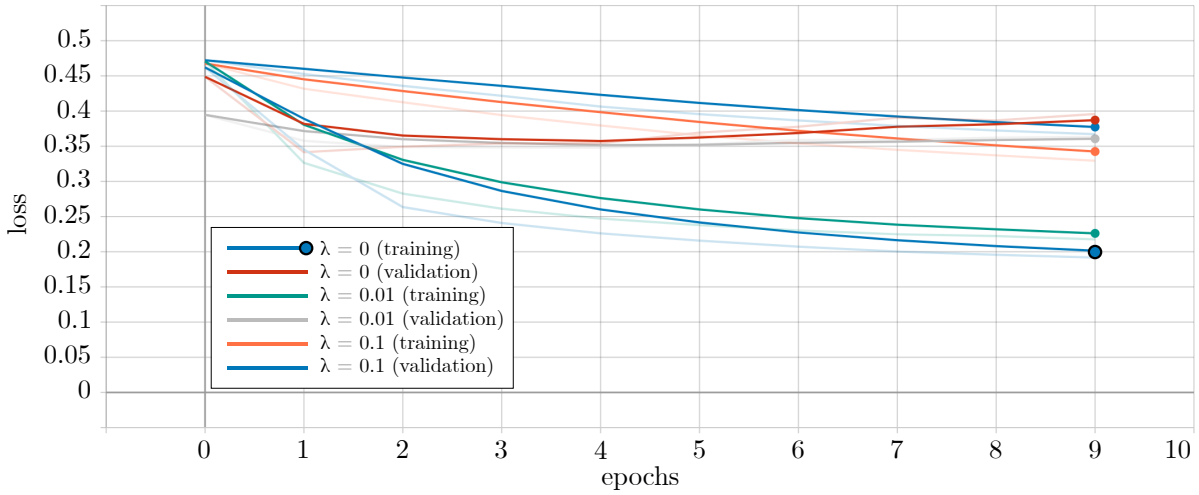


FIGURE 9.13: *Tensorboard accuracy curve comparing the performance of an LSTM with a learning rate of 0.2 and a learning rate of 0.01, respectively.*

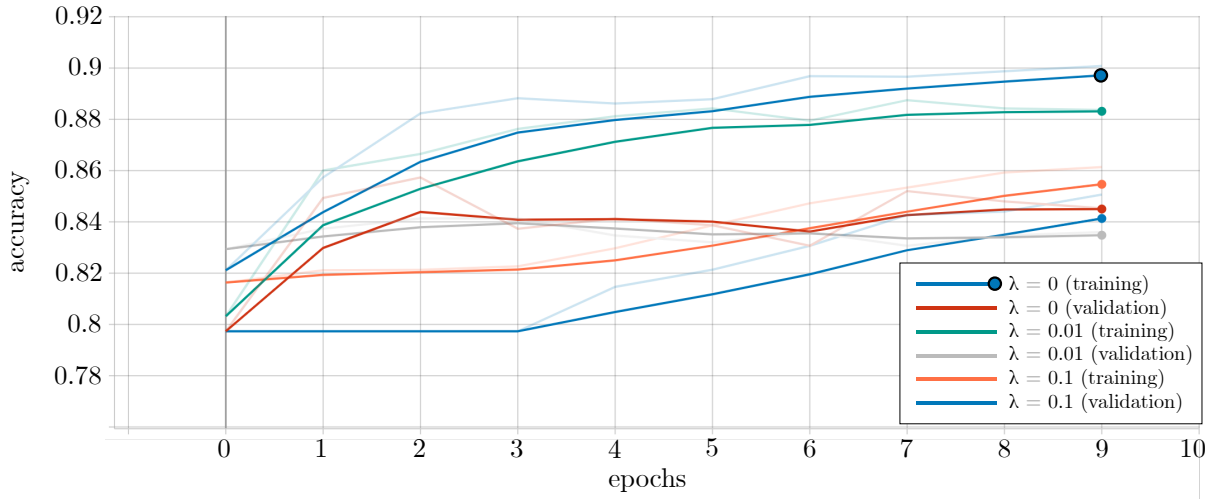
of epochs. Based on the graph, this number was varied between four epochs and eight epochs during the subsequent grid search.

Having established suitable hyperparameter values for the three deep learning algorithms, the selection of the vocabulary size based on the naïve Bayes classifier could be verified using the other algorithms. More specifically, six of the ten vocabulary sizes tested in Figure 9.5 were employed to evaluate the remaining machine learning models. As before, a grid search with 3-fold cross-validation was performed to tune the hyperparameters of the models. The hyperparameter values included in the grid search for each algorithm are shown in Table 9.2. Furthermore, nine separate experiments were conducted for SVM, logistic regression and ANN, one for each possible combination of the document representations term presence, term frequency and TF-IDF, as well as the n-gram ranges unigrams, bigrams and unigrams with bigrams.

The results in Figure 9.16 reflect the AUC score of the best performing model for each algorithm (thus only the best performing result of the nine feature-model pairs tested for SVM, logistic regression and ANN is shown) on a fixed test set as a function of the vocabulary size. In each case, 3-fold cross-validation was performed employing the hyperparameter ranges given in Table 9.2 and selecting the fold with the best accuracy, as per the observation in Figure 9.5 that employing this metric during the grid search, while more computationally efficient, results in an



(a) *Loss*



(b) *Accuracy*

FIGURE 9.14: *Tensorboard graphs for an LSTM with varying regularisation strengths. (a) The loss curves and (b) the accuracy curves of the tested network configurations for the LSTM model generated by the TensorBoard interface are shown.*

AUC score similar to that achieved using the AUC metric. As may be seen in the figure, the AUC scores for all models initially increase significantly with the size of the vocabulary, but the scores then remain relatively constant from a size of approximately 150–200 tokens onwards. These results verify those returned by the experiments in Figure 9.5. A vocabulary size of 250 was therefore retained for all subsequent experiments.

9.2.2 Evaluating the resulting MSTs

Finally, given the reduced selection of hyperparameters and a suitable vocabulary size, the grid search could be performed and the resulting models evaluated and compared. The same document models and n-gram ranges as before were implemented, along with the hyperparameter values given in Table 9.2. In this manner, the effect of each document model could be evaluated, as well as the effect of using longer n-grams both instead of and in combination with shorter n-grams. Consequently, nine *Experiment* classes were instantiated for the naïve Bayes, SVM,

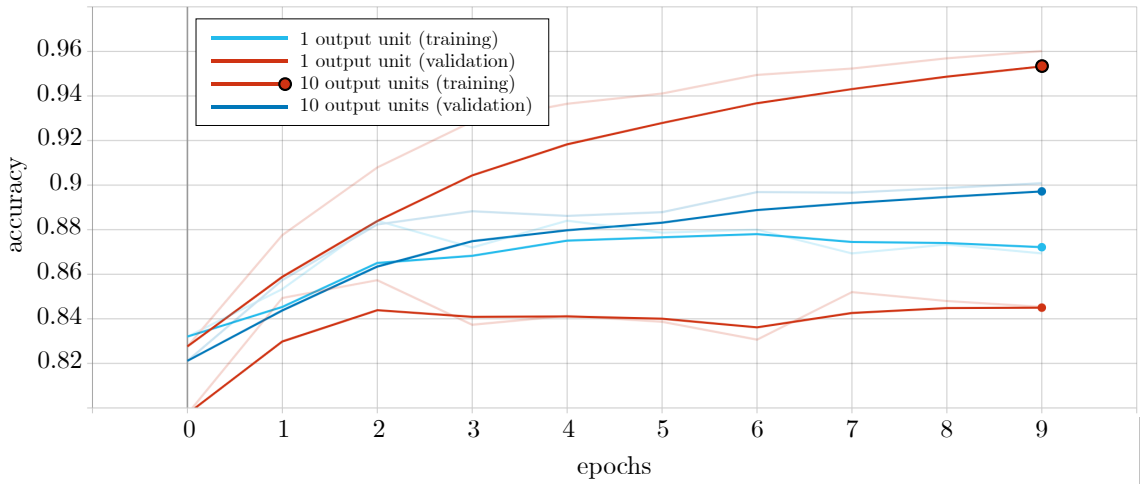


FIGURE 9.15: *Tensorboard accuracy curve comparing the performance of an LSTM with one output unit and ten output units, respectively.*

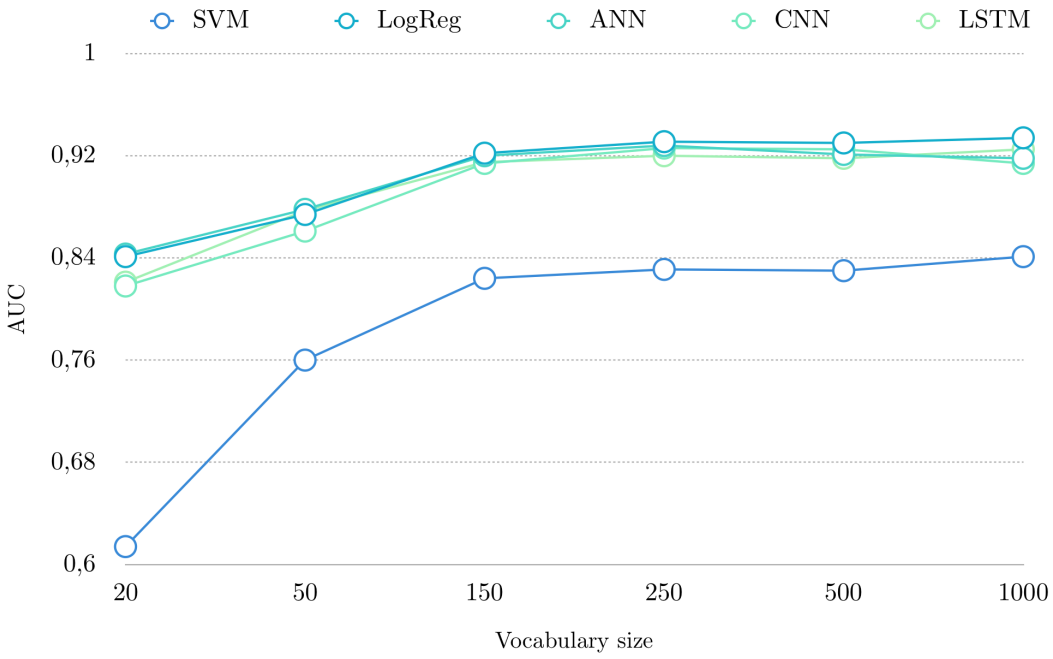


FIGURE 9.16: *The effect of vocabulary size on the AUC value achieved by various machine learning models.*

logistic regression and ANN algorithms, whilst one *Experiment* class was instantiated for CNN, LSTM, Vader, Pattern, Sentiwordnet and Hu and Liu opinion lexicon algorithms. The total number of experiments was therefore forty two $(4 \times 9 + 6)$. In order to account for the variability of the performance of all the models originating from the training data and test data, as well as the variability of the deep learning models originating from the random initialisation of network weights, each experiment was repeated ten times with a different random seed for each replication. The results are therefore presented in the form of box plots, indicating the median performance, as well as the degree of variation around this median. Detailed results of all the replications and experiments can be found in Appendix A.

Algorithm	Hyperparameter	Values
Naïve Bayes	α	0.0001, 0.2, 0.4, 0.6, 0.8, 1
SVM	C	0.1,1,10
	γ	0.01, 0.1, 1
	Kernel	linear, radial, sigmoid, P2
Logistic regression	C	0.01, 0.1, 1, 10
	Solver	SAG, Newton-CG, LBFGS
	Max iterations	100
ANN	Neurons per hidden layer	10,10
	Activation function	ReLU
	Regularisation	ℓ_2
	λ	0.001
	Dropout probability	0
	Batch normalisation	No
	Loss Function	Cross-entropy
	Solver	ADAM
	Number of epochs	10, 12
	Initial learning rate	0.02
	Learning rate decay	0
CNN	Embedding size	10
	(Kernel size, stride, number of filters)	(1 1 20)
	Convolution type	Valid
	Pooling	No
	Activation function	ReLU
	Regularisation	ℓ_2
	λ	0.01
	Batch normalisation	No
	Loss Function	Cross-entropy
	Solver	ADAM
	Number of epochs	10, 12
	Initial learning rate	0.01
	Learning rate decay	0
LSTM	Embedding size	10
	LSTM output size	10
	Dropout probability	0
	Regularisation	None
	Loss Function	Cross-entropy
	Solver	ADAM
	Number of epochs	4, 8
	Initial learning rate	0.01
	Learning rate decay	0

TABLE 9.2: *The hyperparameter values tested during the grid search for all machine learning algorithms.*

The AUC scores achieved by each algorithm are shown in Figure 9.17. Where multiple models were trained by means of the same algorithm (*i.e.* naïve Bayes, logistic regression, SVM and ANN, where the nine variations of the term-document matrix were used as input), the best performing model was selected for each of the ten replications. It is clear from the figure that the machine learning models outperformed the off-the-shelf lexicon-based models by a large margin. Whilst the four lexicon-based models achieved similar median AUC scores in the region 0.59–0.61, constituting performance only marginally better than random guessing, the machine learning models achieved scores in the range of 0.79–0.91, with five of the six models achieving median AUC scores of over 0.89.

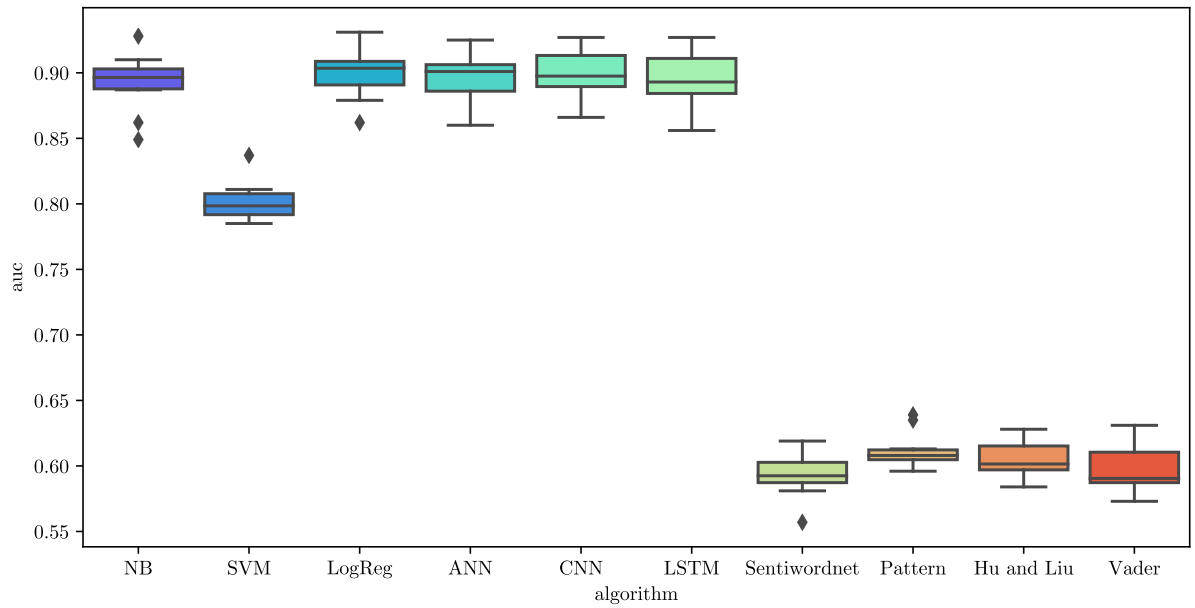


FIGURE 9.17: *Model performance in terms of AUC. The AUC values achieved during ten experimental runs is shown in the form of a box plot for each of the tested models.*

It would appear that SVM is significantly outperformed by the other models in respect of the AUC score. It is important to consider, however, that SVM is the only discrete classifier among the machine learning models. The AUC score is therefore merely an estimate, based on the linearly interpolated approximation of the ROC graph between three single points, as described in §7.2.2. Hence it is possible that this score misrepresents the actual performance of the SVM algorithm to some degree. Whilst the lexicon-based models are also subject to this approximation, the effect of a poor approximation of the ROC curve is unlikely to account for the observed magnitude of the difference in performance of the machine learning models and lexicon-based models.

A zoomed-in version of the graph in Figure 9.17 is shown in Figure 9.18, focusing exclusively on the machine learning models. Logistic regression achieved the highest median AUC score of 0.9010, followed closely by ANN (with a median score of 0.9010), ANN (with a median score of 0.8975), naïve Bayes (with a median score of 0.8930) and LSTM (with a median score of 0.8965), whilst SVM achieved a slightly lower median AUC score of 0.7985. The variability of the scores is larger for the deep learning models than for the '*shallow*' learning models. This may be attributed to the fact that the variability of the latter models originates only from the varying data sets, whilst the deep learning models are also subject to variation in their initial weight parameters. Overall, reserving judgement on SVM, the machine learning models achieve comparable performance on the data set in terms of the AUC score.

The performance achieved by the models in terms of accuracy (and by definition also in terms of the micro-averaged precision, recall and F-measure scores) is shown in Figure 9.19. As in the case of the AUC score, a pronounced distinction between the lexicon-based models and machine learning models is exhibited, with the former achieving median accuracy scores in the range of 39%–47% and the latter achieving median accuracy sores between 82% and 85%. In this case, however, the Sentiwordnet model appears to outperform the other three lexicon-based models by a large margin, followed by the Vader algorithm which, in turn, appears to outperform the remaining two lexicon-based models. It is interesting to note that these models achieved much higher accuracies in other problem domains. In the original paper in which Vader was

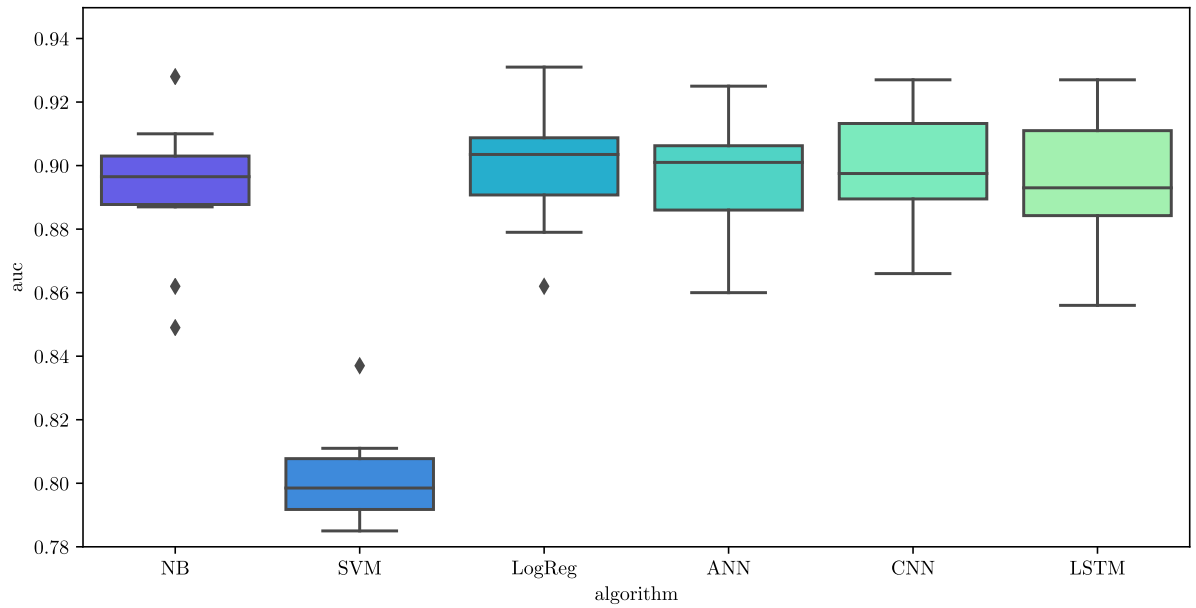


FIGURE 9.18: *The graph of Figure 9.17 zoomed in to show only the machine learning models.*

presented, for instance, an accuracy of 96% was reported for social media texts, whilst 61% and 63% accuracies were reported for movie and product reviews, respectively [131]. These findings support the premise of the ECCO framework, which seeks to facilitate model development rather than apply a specific model in light of the fact that no single model can be guaranteed to outperform all other models in every problem setting. In this manner, a '*good*' sentiment classifier can be constructed for any problem domain.

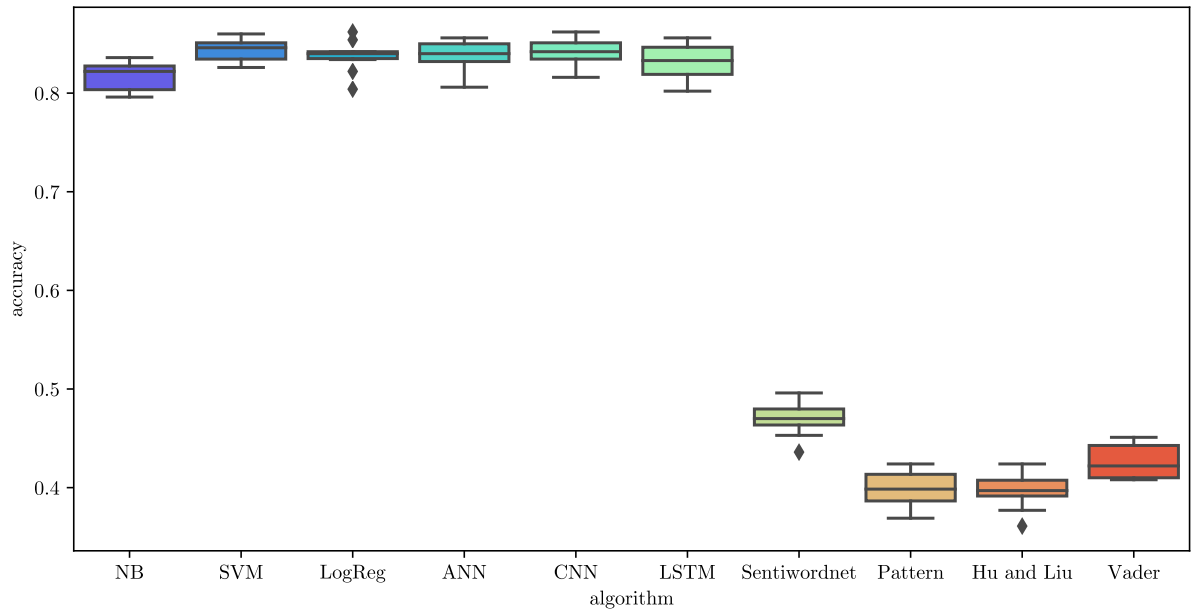


FIGURE 9.19: *Model performance in terms of accuracy. The accuracy values achieved during ten experimental runs is shown in the form of a box plot for each of the tested models.*

In Figure 9.20, the focus is once again shifted towards the machine learning models. In this case, the naïve Bayes algorithm appears to be slightly inferior to the other models, exhibiting

the lower median accuracy of 82.20% and a large variance, with scores ranging between 79.60% and 83.60%. The SVM model, on the other hand, fares favourably in respect of this metric, achieving the highest median accuracy of 84.60%. CNN, ANN and logistic regression follow closely with median accuracies of 84.20%, 84.00% and 84.00% respectively, whilst the LSTM network achieves a median accuracy of 83.30%. Overall, in terms of accuracy, it would appear that all machine models achieve comparable performance, with SVM, CNN, ANN and logistic regression slightly outperforming the remaining machine learning algorithms.

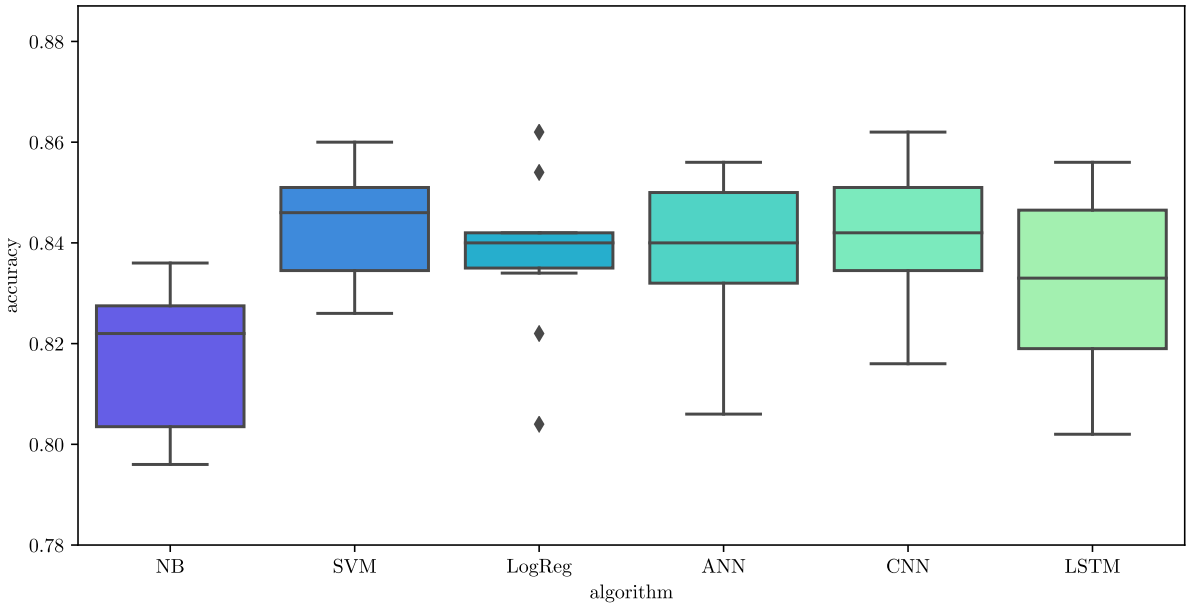
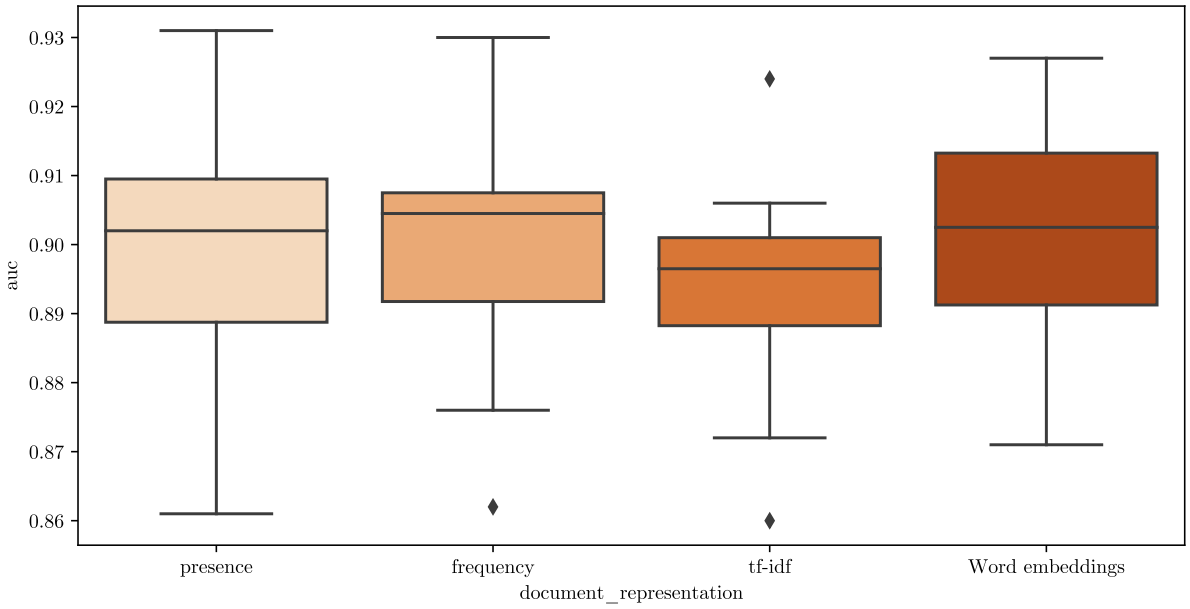


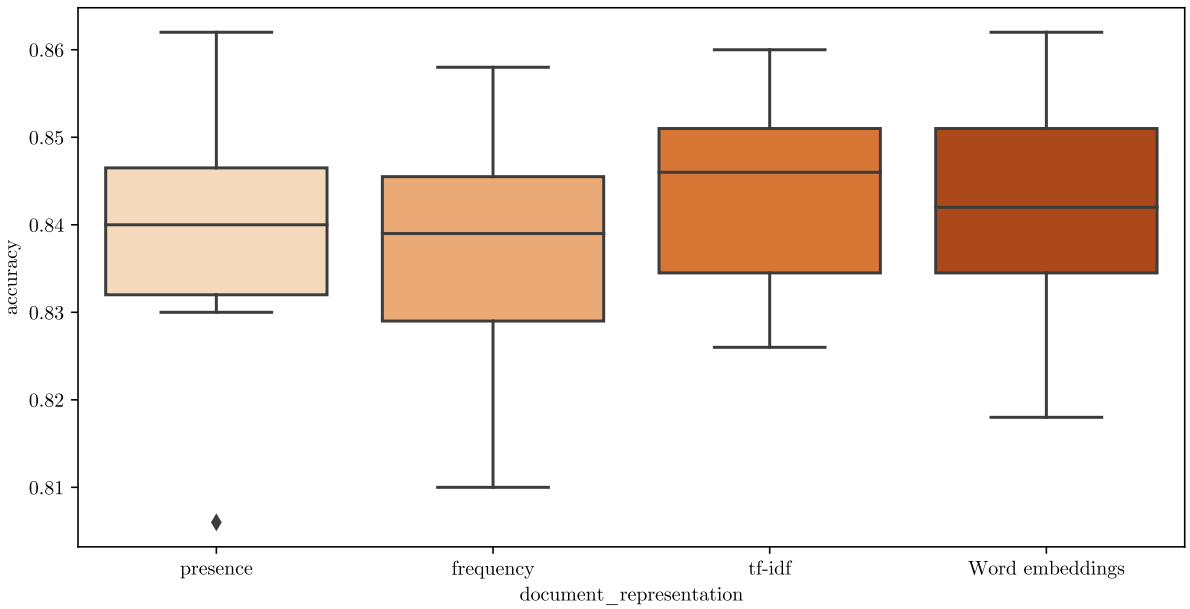
FIGURE 9.20: *The graph of Figure 9.19 zoomed in to show only the machine learning models.*

Similar box plots were constructed to compare the performance of the various document representations. In Figure 9.21(a) and Figure 9.21(b), the AUC scores and accuracy scores are shown as a function of document representation, respectively. As before, the maximum score achieved is selected for each experiment run, resulting in ten data points for each box plot. From these graphs, there seems to be no clear indication that the use of a specific document representation necessarily provides an advantage. Whilst the TF-IDF representation results in a slightly higher median accuracy than the other representations, this document model also achieves the lowest median AUC score. The converse is true for the term frequency document model. The variance in performance is generally lower for the accuracy score than for the AUC score, especially in respect of the term presence document representation. The accuracy scores for all document representations range from 80.60% to 86.20%, whilst the AUC score ranges from 0.860 to 0.931. In either case, the aggregated scores of all experimental runs are similar, with median AUC scores lying between 0.8965 and 0.9045, and median accuracies taking values of 83.9%–84.6%.

A clearer distinction may be observed when the n-gram range is considered. In Figure 9.22, the maximum accuracy scores achieved during each run are displayed in terms of both the document representation (with the exception of word embeddings) and the n-gram ranges of unigrams (indicated as $(1, 1)$ in the figure), bigrams (indicated as $(2, 2)$ in the figure) and unigrams with bigrams (indicated as $(1, 2)$ in the figure). From this figure it is clear that there is a significant decrease in performance when bigrams are used than when unigrams or a combination of unigrams and bigrams are used. The performance of the other two n-gram ranges, on the other hand, are competitive. Looking for common term collocations without regard for the individual words employed in a document therefore causes a decline in performance, whilst adding this



(a) *AUC score*



(b) *Accuracy score*

FIGURE 9.21: *The effect of feature engineering on performance. The highest (a) AUC value and (b) accuracy score achieved by models using various document representations during ten experimental runs is shown in the form of a box plot in each case.*

information to the simple unigram representation does not seem to affect a significant increase in performance in this case. Furthermore, for both unigrams and unigrams with bigrams, the use of term presence and term frequency document models results in a similar performance, whilst the TF-IDF representation performs marginally worse. When only bigrams are employed, on the other hand, there is little distinguishable difference between the document models, save the fact that the TF-IDF representation appears to produce more stable AUC results across the experimental runs.

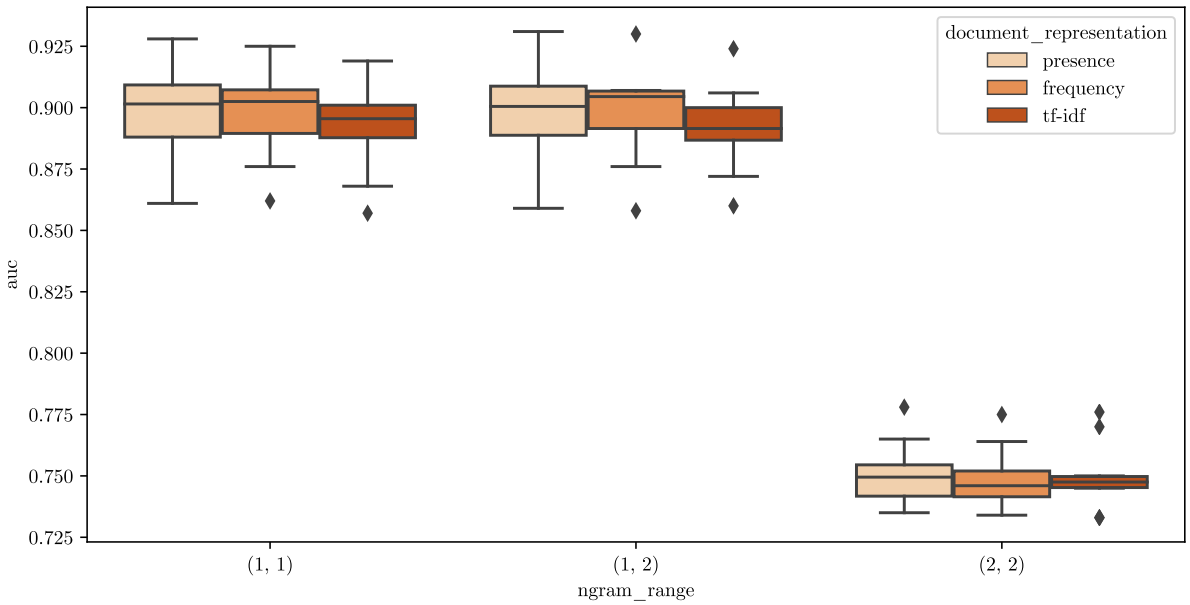


FIGURE 9.22: *The effect of feature engineering on the AUC value. The highest AUC value achieved by models using various document representations and n-gram ranges during ten experimental runs is shown in the form of a box plot in each case.*

In respect of the accuracy score, the same large performance gap is exhibited between the bigrams-only n-gram range and the other two representations, as shown in Figure 9.23. For this metric, a more discernible difference was also found between the different document representations in each case. As shown in the figure, the TF-IDF representation achieves a higher median accuracy for both the unigram and the unigram with bigram n-gram ranges, whilst the term presence model appears to perform slightly better for the bigram representation. Considering the importance of a word based on the frequency of its usage within a document and within the corpus as a whole could thus be beneficial for sentiment classification. It is likely that this pattern is not exhibited in the bigrams only case due to fewer appearances of each bigram in the corpus than individual unigrams.

The patterns exhibited in respect of the document representations and n-gram ranges in Figures 9.22 and 9.23 were similar when considered separately for each algorithm. This can be verified by referring to the detailed experimental results in Appendix A.

In summary, the experiments showed that the off-the-shelf lexicon-based methods were significantly outperformed by the machine learning models developed according to the ECCO framework. The machine learning models achieved overall competitive performances, with the performance of SVM and naïve Bayes proving slightly inferior in terms of the AUC score and accuracy, respectively. Based on both metrics, the top three performing algorithms were logistic regression, CNN and ANN in no discernible order. With respect to the input features, there is a clear decline in performance when bigrams are used in isolation without including unigrams

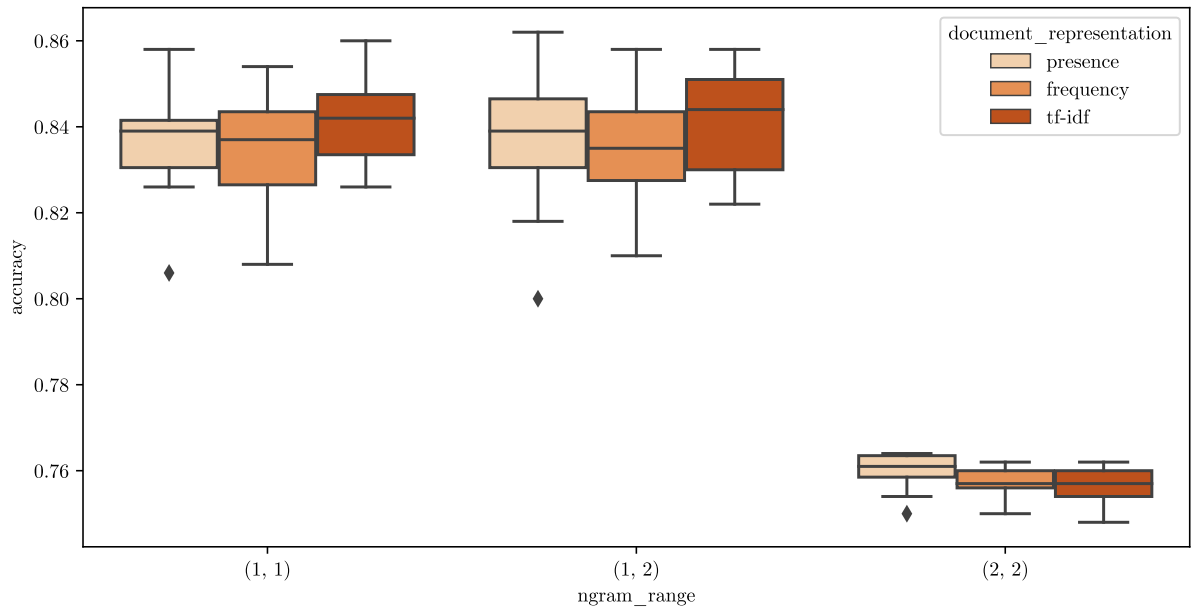


FIGURE 9.23: *The effect of feature engineering on accuracy. The highest accuracy value achieved by models using various document representations and n-gram ranges during ten experimental runs is shown in the form of a box plot in each case.*

in the feature vector. Differences in performance based on document models were more difficult to identify. It would appear that the use of the TF-IDF representation had a positive effect on the accuracy score, but a negative effect on the AUC score (particularly for unigrams and unigrams with bigrams), whilst word embeddings achieved competitive performance with the best performing document representations in respect of both metrics. These differences are, however, marginal.

The classification results of the third-party software described in the previous chapter were also evaluated in respect of the labelled data. Unfortunately, the software's ratings were not available for all of the data. The evaluation was therefore carried out in respect of 2 486 (99.44%) of the 2 500 labelled reviews. The confusion matrix illustrating the results of the classification is shown in Figure 9.24.

As is evident from the figure, the software often fails to distinguish between the negative and the neutral class, resulting in an accuracy score of 59.05% and a micro-weighted AUC score of 0.6602. Compared with the lexicon-based models evaluated during the case study, which achieved median AUC scores between 0.60 and 0.62 (with a maximum score of 0.649) and median accuracies between 39% and 48% (with a maximum score of 50.80%), the software thus fares favourably. In comparison with the machine learning models developed by means of the ECCO system, however, the software does not achieve competitive results. The median scores for the AUC and accuracy score range between 0.79–0.91 and 82%–85% for these models, respectively. Since the software does not make use of any annotated training data from the industry partner, it likely makes use of either a lexicon-based model or a machine learning model that has been pre-trained in respect of other annotated data. The results achieved by this software in respect of the data from the industry partner further highlight the problem of applying a model developed within one specific context to data originating within another context. Off-the-shelf models or readily available software by a third-party vendor may produce adequate results if the application context is sufficiently similar to the development context. If

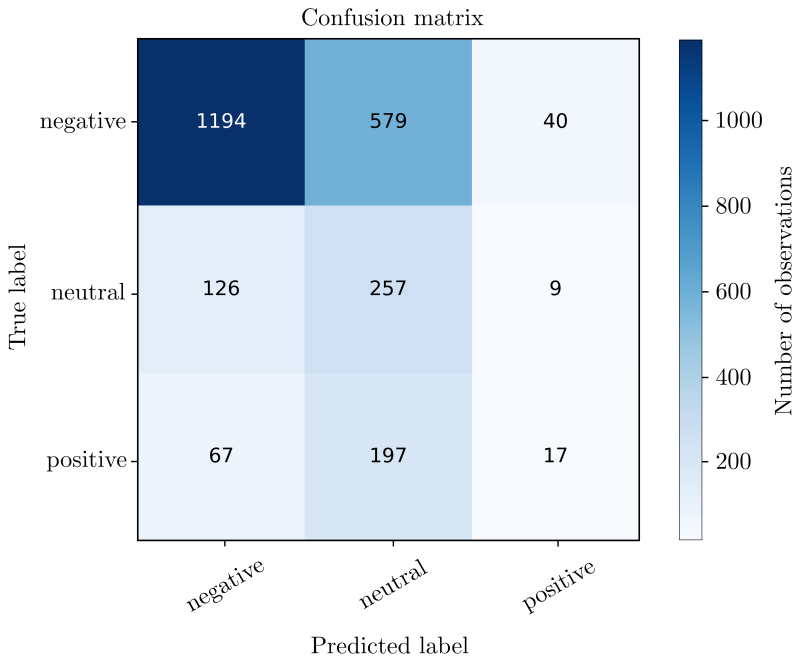


FIGURE 9.24: *The classification results of the third-party sentiment analysis software in the form of a confusion matrix.*

this is not the case, however, a significant improvement in performance may be achievable by developing context-specific models according to the process outlined in the ECCO framework.

9.2.3 Generating ensembles of selected MSTs

During the next phase of the model development process, it was investigated whether combining several models in an ensemble could further improve performance. To this end, all available ensemble configurations implemented in the ECCO system were compared. More specifically, an experiment was performed for each possible combination of the output type employed in the ensemble (discrete or scoring outputs) and the method of output combination (simple voting, weighted voting and meta-learning), resulting in six (2×3) experiments.

As mentioned in §3.4.5, however, the task of selecting suitable base learners for an ensemble is non-trivial. In this particular case, in which there are forty two different models that may be included, $\sum_{K=1}^{42} \binom{42}{K} = 2^{42} - 1 \approx 4$ trillion options exist to form a set of base learners. Based on the literature review on ensemble pruning in §3.4.5, two approaches were employed in this case study. The first is a simple greedy approach, in which the K base learners with the largest mean test accuracy across all ten experimental runs are selected. Here, the value of K was arbitrarily set to five.

The second approach entails selecting a set of base learners for which an original composite measure is maximised which represents the *favourability* F of a set of base learners. In this context, a favourable composition of base learners is as accurate and as diverse as possible, as per the relevant literature. Furthermore, there should be a bias towards including a larger number of base learners, in line with Condorcet's Jury Theorem (see §3.4). Ensemble favourability is therefore expressed as

$$F = \lambda_1 A + \lambda_2 D + (1 - \lambda_1 - \lambda_2)\frac{K}{N},$$

where A is the average accuracy of the base learners included, D is a measure of diversity of the ensemble, K/N is the proportion of available models selected as base learners, and $\lambda_1 \in [0, 1]$, $\lambda_2 \in [0, 1]$ and $(1 - \lambda_1 - \lambda_2)$ are weights representing the relative importance of base learner accuracy, ensemble diversity and ensemble size, respectively.

Motivated by the findings of Whalen and Pandey [338], namely that employing only one base learner per learning algorithm achieved consistently good results, as well as the generally observed trend that employing different feature sets to train different base learners leads to superior results (see §3.4), ensemble diversity was quantified in this dissertation as the degree of differentiation between base learners in respect of the learning algorithm and feature set employed. More specifically, in this case study, each candidate base learner i was assigned a vector $\boldsymbol{X}_i = \begin{bmatrix} X_{i1} & X_{i2} & X_{i3} & X_{i4} \end{bmatrix}$, where X_{i1} represents the approach to sentiment classification employed (*i.e.* lexicon-based or machine learning), X_{i2} represents the specific algorithm employed (*e.g.* SVM or SentiWordNet), and X_{i3} and X_{i4} are the first and second feature dimension, respectively. The first feature dimension constitutes the document representation for base learners employing the bag of words model as input (*i.e.* term presence, term frequency or the TF-IDF weighting) and the types of features employed by the remaining algorithms (*i.e.* word embeddings for the CNN and LSTM models and affect words for the lexicon-based methods). The second feature dimension denotes the n-gram range for bag of words models, the type of word embedding employed (*i.e.* pre-trained or end-to-end) or the specific sentiment lexicon employed (*e.g.* the Hu and Liu opinion lexicon or SentiWordNet). The diversity of a set of base learners can then be quantified by determining the degree of diversity between each pair of base learners in the set across all four dimensions, and then normalising the sum of these values for all pairs in the ensemble.

In contrast to comparing the prediction errors of different base learners in respect of different classes, the proposed approach to quantifying diversity may be viewed as a *prediction* of diversity in classification. The selection of meta-features in respect of which to define diversity is therefore critical in ensuring that this measure of diversity is helpful to the ensemble model. The computational expense of this approach is, however, comparatively low, since individual predictions of candidate base learners do not need to be stored or compared directly. In fact, this measure of diversity can be computed prior to deploying candidate models. If some measure of accuracy is already available for candidate base learners, such as historical performance in respect of similar tasks, this ensemble selection approach avoids the costly re-training of candidate models that are not ultimately selected.

Mathematically, ensemble diversity may be expressed as

$$D = \frac{\sum_{a=1}^{N} \sum_{i=a+1}^{N} \sum_{j=1}^{Z} I(X_{ij} = X_{aj}) \ell_a}{\binom{K}{2} Z}, \tag{9.1}$$

where N is the number of candidate base learners, Z is the number of dimensions across which diversity is measured, ℓ_a is a binary variable indicating whether or not candidate base learner a is included in the ensemble, $K = \sum_{a=1}^{N} \ell_a$ is the number of base learners chosen and I is the indicator function, which equals one if the statement in its argument is *true*, or zero, otherwise. If every pair of base learners included in the ensemble is diverse in respect of all dimensions considered, then $D = 1$. Moreover, if all pairs of ensembles are equal across all dimensions considered, then $D = 0$. The accuracy term may be expressed in terms of ℓ as

$$\frac{\sum_{a=1}^{N} \alpha_a \ell_a}{K}.$$

The most *favourable* set of base learners may then be selected from the available candidates by solving the optimisation problem

$$\max F = \lambda_1 \frac{\sum_{a=1}^{N} \alpha_a \ell_a}{K} + \lambda_2 \frac{\sum_{a=1}^{N} \sum_{i=a+1}^{N} \sum_{j=1}^{Z} I(X_{ij} = X_{aj}) \ell_a}{\binom{K}{2} Z} + (1 - \lambda_1 - \lambda_2) \frac{K}{N} \tag{9.2}$$

$$\text{subject to } K = \sum_{a=1}^{N} \ell_a. \tag{9.3}$$

The parameters λ_1 and λ_2 were configured by considering a small *test problem* with three candidate base learners and tuned such that the desired outcome was achieved in a number of cases. More specifically, if all models are equal across all four dimensions ($D = 0$), the single model with the highest accuracy should be selected if the difference between accuracies is large. If the difference between individual accuracies is relatively small, however, larger numbers of base learners should be preferred. If the accuracies achieved by the three candidate models are 0.6, 0.6 and 0.9, for instance, the model with the highest accuracy should be selected. If, on the other hand, the candidate models were to achieve accuracies of 0.8, 0.8, and 0.9, respectively, all three candidates should be included in the ensemble. Accordingly, if all candidate base learners are perfectly diverse and have the same accuracy ($D = 1$ and the values of A are equal for all possible subsets), all candidate base learners should be selected in order to maximise the ensemble size. Finally, if one model is entirely diverse from both other candidates, then that model should be selected along with the other candidate with the highest accuracy in order to maximise diversity. The selected parameter values that sufficiently satisfy these conditions were found to be $\lambda_1 = \lambda_2 = 0.45$.

The resulting optimisation model was implemented in an `MS Excel` spreadsheet and solved using a genetic algorithm (see §2.4.4) with default hyperparameters, namely a population size of 100, a mutation rate of 0.75 and a convergence tolerance of 0.001. The resulting selection of base learners for the case study data was the set of the best-performing models in each of the following categories: Machine learning models employing the bag-of-words representation as input, deep learning models trained end-to-end with a word embedding matrix, and lexicon-based models. More specifically, logistic regression trained in respect of a bag of words model with a term presence document representation and an n-gram range of $(1, 2)$, the CNN model and SentiWordNet were selected.

Since the lexicon-based models performed significantly worse than the machine learning models, the optimisation model was also applied to a candidate pool comprising only the machine learning models. In this case, the selected models were the same two machine learning models as those selected from the full pool of candidate models, along with an ANN model taking a bag of words model with a term frequency document representation and an n-gram range of $(1, 1)$ as input. By comparing the performance of the ensembles formed by applying the model-based selection to both sets of candidate base learners, the effectiveness of the ensemble favourability model could be evaluated both in the case of large discrepancies in individual candidate accuracies with high potential diversity, and lower differences in accuracies with more similar models (lower diversity). In the remainder of this dissertation, these variants of the model-based ensemble pruning approach are referred to as the *model* and the *model ML* approach, respectively.

In the greedy approach, all selected base learners had been trained by means of the logistic regression algorithm. The feature sets employed were a bag of words with a unigram term presence, unigram with bigram term presence, unigram term frequency, unigram with bigram term frequency and unigram TF-IDF representation, respectively. A summary of the selected

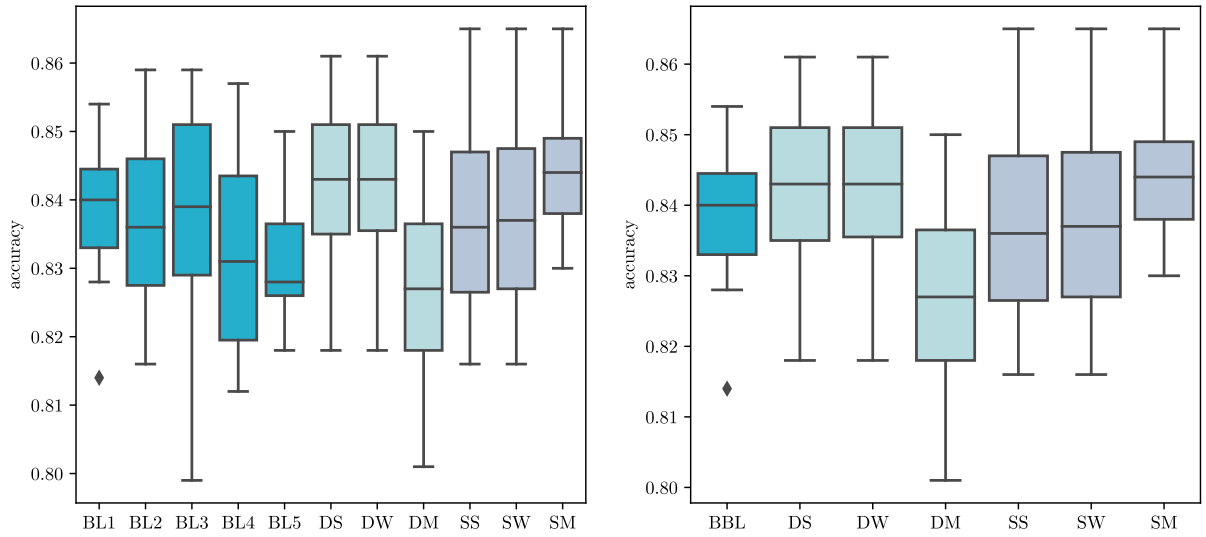
ensemble configurations and their associated diversity, accuracy and favourability metrics are shown in Table 9.3.

Selection approach	K	Diversity	Accuracy	Favourability
Greedy (Top 5)	5	0.3500	0.8529	0.5532
Model (All candidates)	3	0.9167	0.7199	0.7436
Model ML (ML only)	3	0.7500	0.8409	0.7238

TABLE 9.3: *The three sets of base learners evaluated during the case study.*

As before, to account for variability in training and testing data, each of the eighteen experiments (six experiments for each of the three base learner sets) was repeated ten times, in respect of the same random splits employed for the individual model experiments. All reported results reflect performance in respect of the test data. Detailed results may be found in Appendix D.

The results for the greedy ensemble selection approach are shown in the form of box plots in Figure 9.25. More specifically, the relative performances of all *base learners* (BLs) and ensemble configurations is shown in Figure 9.25(a), whilst only the performance of the *best base learner* (BBL) is shown in Figure 9.25(b) along with the performances of the ensemble methods. The results for ensembles using each possible combination of output type and combination method are included separately, namely *discrete output with simple voting* (DS), *discrete output with weighted voting* (DW), *discrete output with meta-learning* (DM), and the equivalent configurations using the scoring output (SS, SW and SM).



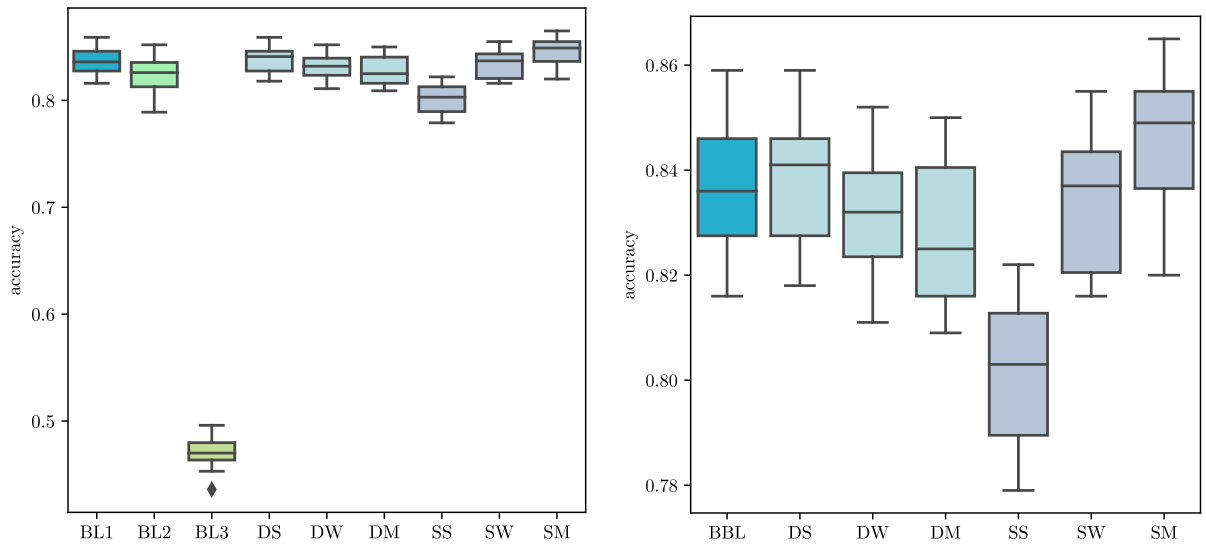
(a) *All base learners* (b) *Best base learner*

FIGURE 9.25: *The accuracies achieved by base learners and various ensemble configurations for the greedy ensemble selection approach.*

From Figure 9.25, it is evident that three of the six ensemble configurations, namely DS, DW and SM, achieved median performances greater than that of the best base learner in terms of median classification accuracy. Furthermore, the remaining two ensembles employing scoring outputs matched or outperformed three of the five base learners. The discrete meta-learning approach, on the other hand, performed slightly worse than the poorest base learner. The best-performing

ensemble model achieved a median accuracy of 84.39% — A mere 0.5% relative improvement over the 84% median accuracy achieved by the best base learner.

Similar graphs are shown for the model ensemble selection method in Figure 9.26. As illustrated in Figure 9.26(a), the range of accuracies achieved by the base learners selected by means of the favourability model is much larger than that of the base learners selected by means of the greedy approach. Nevertheless, three of the six ensemble configurations outperformed the best base learner. The aggregation of scoring outputs by means of simple voting performed comparatively poorly in respect of this set of base learners. This is likely due to the fact that scoring outputs were not available for the discrete lexicon-based classifier. A binary vector indicating the class chosen by the classifier was therefore employed instead, misleadingly asserting a 100% *confidence* or score of the classifier for the predicted class label. The voting ensemble was therefore inclined to give a larger effective weight to the poorest performing base learner. The meta-learning approach, on the other hand, appeared to effectively combine weak and strong base learners to form an even stronger ensemble model. The interquartile range of the accuracies achieved by the SM ensemble configuration is [83.65%, 85.50%], compared to the interquartile range of [82.75%, 84.60%] presented by the best base learner. The median accuracy of this ensemble was 84.90%, a 1.5% relative improvement over the 83.60% median accuracy achieved by the best base learner.

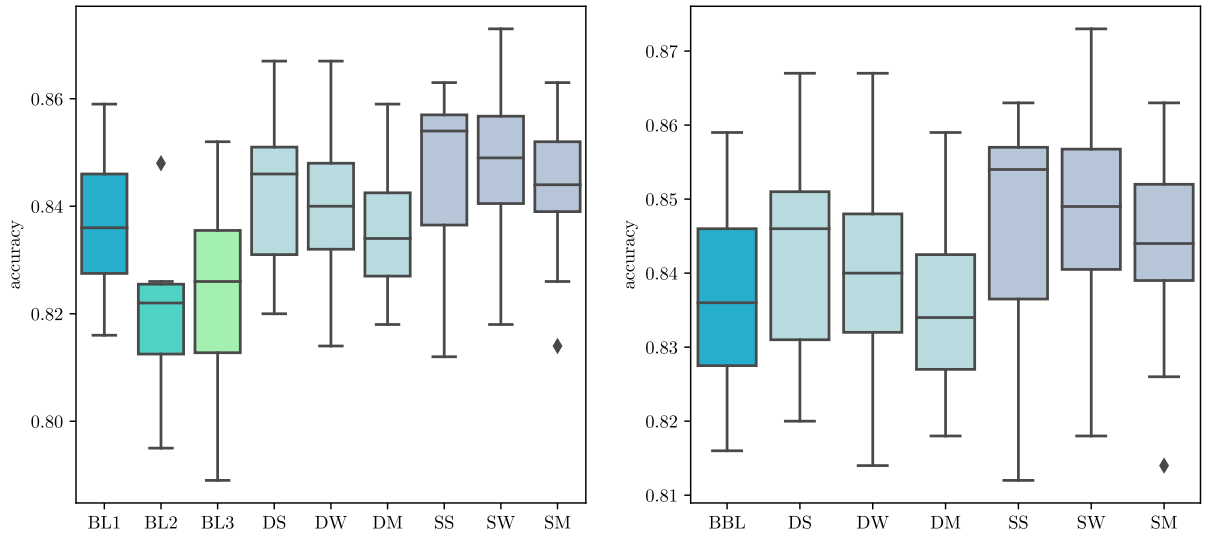


(a) *All base learners* (b) *Best base learner*

FIGURE 9.26: *The accuracy achieved by various ensemble configurations with base learners selected by means of the proposed favourability model.*

Finally, the results achieved by the ensemble employing model ML ensemble pruning are shown in Figure 9.27. In this case, five out of the six evaluated ensemble configurations outperformed the best base classifier, with the sixth configuration, a meta-learner trained in respect of discrete base learner outputs, performing similarly to the best base learner. Although SVM is a discrete classifier, an approximation of the classifier's probability scores for each class was employed for the scoring ensemble configurations. This approximation, which is built into the SVM classifier of the `Scikit-learn` library [236], was computed by training a logistic regression classifier in respect of the outputs of the SVM model and then employing the probabilities of this classifier as a proxy. The problem encountered in the case of the lexicon-based model in the previous

ensemble configurations was therefore circumvented[2]. The best-performing ensemble model for this set of base learners was the simple scoring ensemble, with a median accuracy of 85.4%, a relative improvement of 2.2% over the 84.6% median accuracy achieved by the best base learner.



(a) *All base learners* (b) *Best base learner*

FIGURE 9.27: *The accuracy achieved by various ensemble configurations with base learners selected by means of the proposed favourability model applied only to the machine learning models.*

A summary of the results achieved by all eighteen ensemble experiments in respect of accuracy and the AUC score is shown in Figures 9.28 and 9.29, respectively. The dashed black lines in the figures denote the median performance of the best-performing individual model. From these figures, it is clear that, although the discrete output type achieves favourable performance when employed with sufficiently accurate base learners (as in the case of the greedy and model ML selection), the AUC scores of the resulting ensemble models are significantly lower than the baseline set by the best individual model. The combination of scoring outputs by means of meta-learning, on the other hand, performs well across all three selection mechanisms, and is able to effectively leverage the high level of base learner diversity in the model selection approach in order to achieve visibly better results than the baseline and the greedy approach. Overall, the best results are achieved by the ensemble model formed by means of a simple voting combination of the scoring outputs of base learners selected *via* the model ML approach. These results suggest that there is promise in selecting base learner configurations by means of the proposed ensemble favourability metric. If a meta-learning combination approach is employed, the equal weighting of ensemble diversity and accuracy adopted in this case study appears to yield favourable results. In the case of simple aggregation, on the other hand, accuracy appears to be a more important metric, since significantly better performance is achieved by means of the model ML selection approach, which is equivalent in this case study to imposing a minimum accuracy constraint on the base learners.

[2]This approach was not adopted when calculating the AUC scores of the SVM classifier, since the resulting approximate probabilities are not directly employed by the classifier and may even result in different classifications [236]. They are, therefore, not necessarily a good representation of the classifier's ability to separate classes. Furthermore, training an additional model to compute probabilities would further aggravate the high computational cost associated with the SVM classifier. For the purposes of the ensemble modelling, however, this approximation was deemed useful.

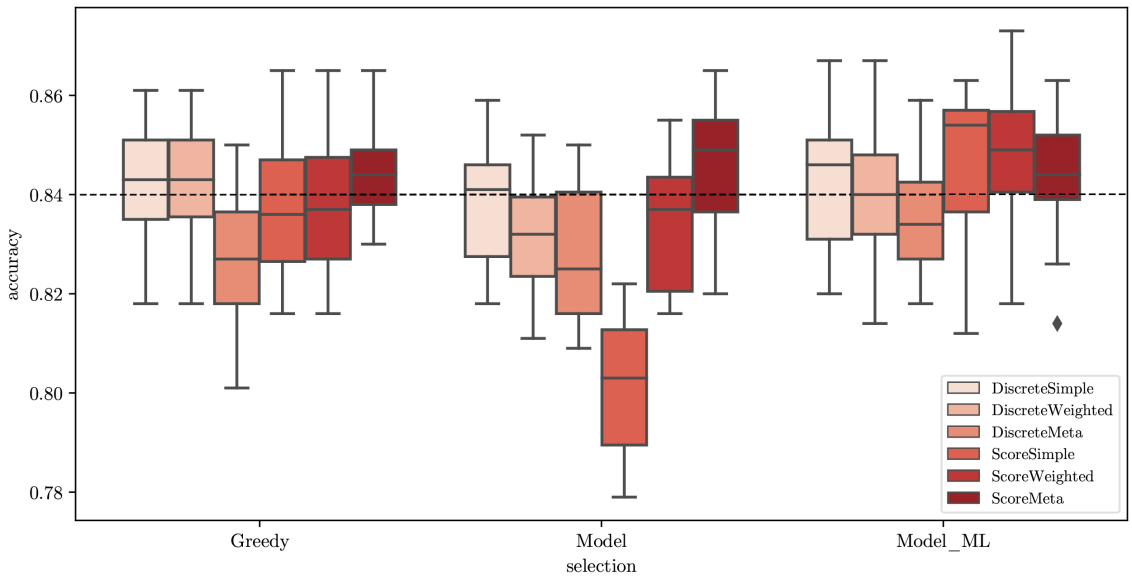


FIGURE 9.28: *Box plots of the accuracy achieved by ensembles in respect of the adopted ensemble pruning approach, output type and combination method employed during ten experimental runs. The median accuracy achieved over ten experimental runs of the best base learner is denoted by the dashed black line.*

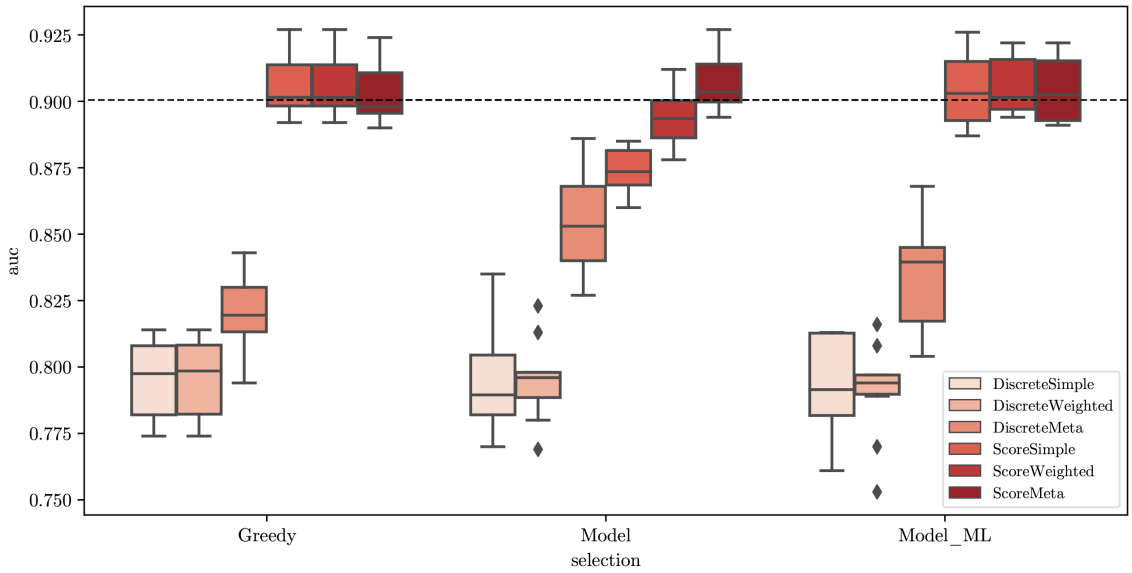
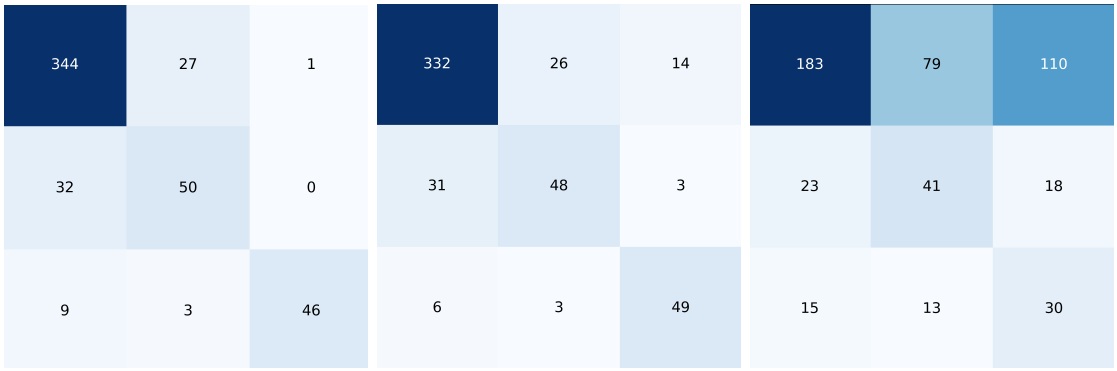
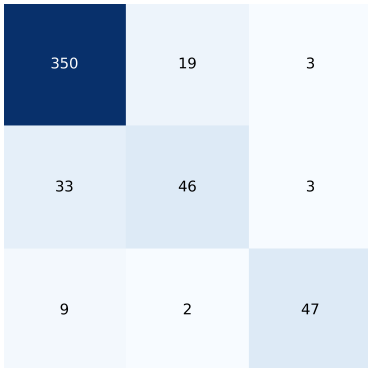


FIGURE 9.29: *Box plots of the AUC scores achieved by ensembles in respect of the adopted ensemble pruning approach, output type and combination method employed during ten experimental runs. The median AUC score achieved over ten experimental runs of the best base learner is denoted by the dashed black line.*

The correlation between the proposed diversity metric in (9.1) and more traditional notions of model diversity related to the errors made in respect of different observations was explored by means of confusion matrices, as shown in the output of the ECCO system in Figure 7.15. The confusion matrices of the most diverse set of base learners selected according to this metric are shown in Figures 9.30(a)–9.30(c) for the first repetition of the experiment. Base learners 1 and 2 both appear to be primarily *confused* between negative and neutral observations, with both models most frequently classifying *neutral* observations as *negative*, or *negative* observations as *neutral*. Base learner 2, furthermore, also misclassifies about half as many *negative* observations as *positive*. Overall, however, both classifiers appear to be fairly accurate, with the largest number of observations appearing on the diagonal of the matrix. Base learner 3, on the other hand, presents a much less accurate pattern of classifications, with many *negative* observations classified as *positive* or *neutral*, as indicated by the darker shading of the first row of the matrix.



(a) *Base learner 1* (b) *Base learner 2* (c) *Base learner 3*



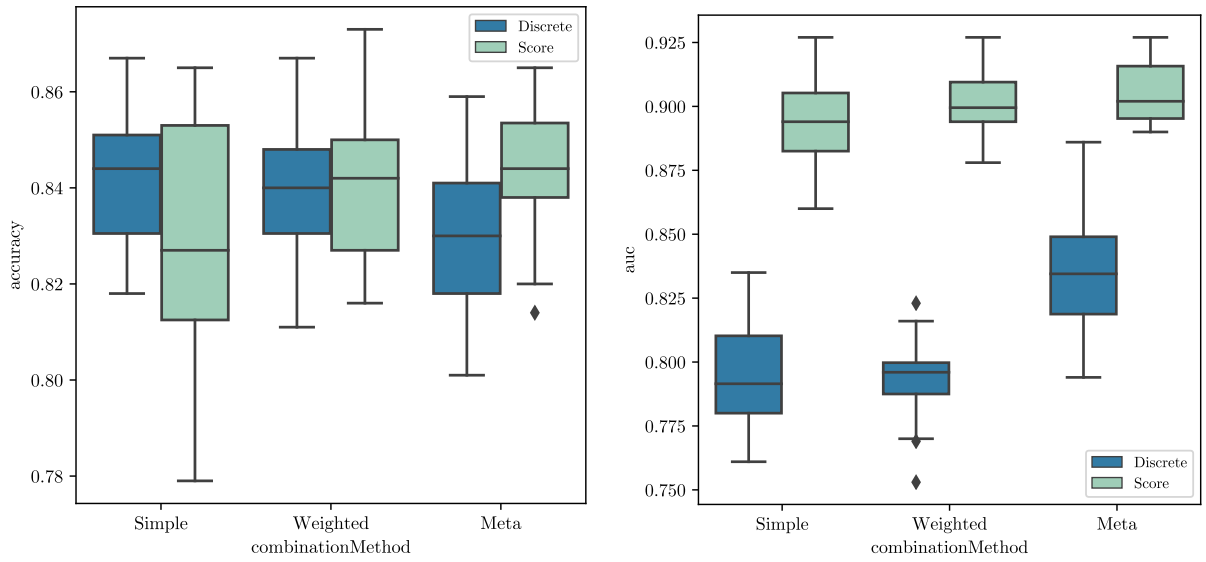
(d) *SM ensemble*

FIGURE 9.30: *Confusion matrices illustrating the errors made when classifying observations of different classes for the base learners chosen by means of the proposed favourability model in (a)–(c) and the ensemble model combining the scoring outputs of these base learners by means of meta-learning in (d). The rows in the matrix indicate the true labels negative, neutral and positive of the observations, respectively, whilst the columns denote the classifications returned by the model in the same order. These results correspond to the first of ten experimental runs.*

Hence there appears to be a considerable level of diversity in the classification errors of the base learners, particularly between the first and third, and the second and third base learners. The ensemble combining the outputs of these base learners by means of a meta-learning approach effectively combined the strengths of each of the base learners. As shown in Figure 9.30(d), the ensemble correctly classifies more negative observations than any base learner, with fewer such observations misclassified as neutral and a higher overall accuracy.

Predicting a diversity in prediction errors by means of the proposed diversity metric has distinct advantages. More specifically, diversity may be computed in this way prior to deploying individual models and without having to store individual prediction results to compute more complex measures of diversity. Care must be taken, however, when selecting the dimensions according to which diversity is measured, in order to ensure that these are appropriate predictors of differing model strengths.

Finally, a comparison of the performance of different output types and combination methods employed in an ensemble is shown in Figure 9.31 in respect of both accuracy and the AUC score. These figures corroborate the previous findings that the combination of discrete outputs results in inferior results in respect of the AUC score, and that the combination of scoring outputs by means of meta-learning produces the most consistently good results in respect of both metrics.



(a) *Accuracy* (b) *AUC*

FIGURE 9.31: *The (a) accuracy and (b) AUC score of ensemble models in respect of the output type and combination method employed during ten experimental runs.*

9.3 Analysis of the model results

Based on the results detailed in §9.2, the CNN model was selected to classify the sentiment polarities of the case study data[3]. The hyperparameter values of this model are given in Table 9.2, where the number of training epochs were set to 12 by the 3-fold cross-validated grid search. According to this model, 75.10% of the free-form text responses submitted by clients who rated[4] their experience at the bank as a 2 or 3 have a negative sentiment polarity, as shown in Figure 9.32. Furthermore, 15.50% of responses bear no sentiment, whilst 9.39% have a positive sentiment polarity.

The most frequent terms in the reviews of the *negative* sentiment class include the general terms *bank* and *money*, which reveal little about the grievances of customers. The term *loan* is also

[3]Although the ensemble models achieved superior results over their individual constituent models, the 2% relative increase in accuracy was not deemed large enough to significantly alter the classification results.

[4]Only 10 053 (97%) of the 10 354 available responses were included in this analysis due to the removal of reviews that did not have a corresponding entry in the *Client demographics* data set.

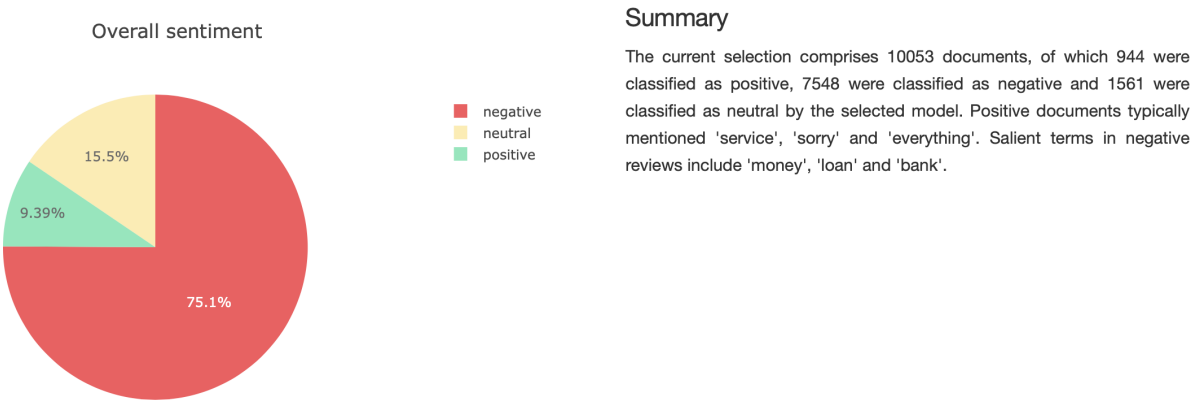


FIGURE 9.32: *The sentiment class distribution and textual summary of the case study data*

frequently used, which may indicate that many customers have complaints pertaining to personal loans. One of the most frequently used terms in *positive* reviews is *sorry*, reinforcing the idea that these customers may have misunderstood the rating scale or accidentally submitted the wrong rating, as was observed by the data science department of the industry partner. These reviews also typically mention *service* and *everything*, which may indicate that the customers who submitted these responses are, in fact, satisfied with the bank as a whole, especially with the service. A more detailed illustration of the word usage per sentiment class is given in the form of word cloud representations in Figure 9.33.

Examining the word cloud for the *positive* sentiment class in Figure 9.33(a) reveals common phrases such as "*sorry [I] meant*," "*made mistake*" and "*wanted [to] say*," which underscore the notion that the rating scale was misinterpreted by these customers. Other common sentiments are reflected in the phrases "*everything fine*," "*nothing wrong*," and "*good service*." Customers in this review category therefore seem to be satisfied overall with the bank and its products and services.

The word cloud of the *neutral* responses in Figure 9.33(b) is more difficult to interpret. The most common terms appear to be *loan*, *Ok* and *need*. The phrase "*need loan*" is also a frequent collocation. Responses in this category thus often seem to refer to loan requests. Other phrases, such as "*smile*," "*funeral cover*" and "*buy airtime*," however, suggest a wide variety of topics. The following five random samples of reviews in this sentiment class were drawn by means of the associated function of the ECCO system in order to gain a better understanding of the subject matter:

(i) "*You no better I dont*,"

(ii) "*Ok*,"

(iii) "*Greetings and smile please*,"

(iv) "*Triple repayment*," and

(v) "*Open the business accounts*."

Once again, a variety of topics is discussed without a clear sentiment orientation and with little actionable information. Those comments classified as *neutral* by the model therefore appear to constitute short answers that bear little information or sentiment and are often difficult to contextualise.

(a) *Positive* (b) *Neutral*

(c) *Negative* (d) *Negative, with certain terms excluded*

FIGURE 9.33: *Word cloud representations of the reviews in each sentiment class.*

Finally, the word cloud illustrating the word usage amongst *negative* reviews in Figure 9.33(c) is dominated by generic terms, such as *bank*, *money*, *account* and the name of the bank (*_bankName*). In order to better analyse the meaningful words in this context, these words were excluded from the word cloud along with other non-informative words, such as *get*, *want* and *I'm*, resulting in the word cloud shown in Figure 9.33(d). From this figure it is now evident that the most important keywords mentioned in negative reviews are *loan* and *ATM*, and that many customers also made reference to *help* in their negative response. Other frequent terms that may bear some information include *time*, *app*, *service*, *staff* and *card*. This elementary analysis sheds some light onto the contents of customer complaints. An LDA topic analysis was

also performed in order to gain a deeper insight into these contents before the customers' most important points of concern were further explored.

After some experimentation, an LDA topic model with five topics was fit to the case study data over two iterations (or *passes* through the data set). The resulting two-dimensional projection of the topics is shown in Figure 9.34, along with the thirty most *salient* words in the corpus. From the figure, it may be deduced that the five discovered topics are relatively well separated with the exception of Topic 1 and Topic 2, which overlap considerably. Topics 1–4 and 5 are distributed along the first principal component (the horizontal axis), while exhibiting similar values for the second principal component (the vertical axis). Topic 4, however, is clearly distinguished from the remaining topics by its value for the second principal component. Furthermore, Topics 1–3 appear in the corpus with relatively equal frequencies, whilst Topics 4 and 5 appear less frequently.

Based on both the frequency with which each of the most *salient* terms in the corpus were observed in a given topic and the *relevance* of words to a given topic, several important keywords could be associated with each topic. These associations are given in Table 9.4. It was furthermore deduced, based on the results in the table and the relative frequencies of these terms in each of the topics, that Topic 1 is concerned with general inquiries related to money, rates and accounts at the bank. Topic 2, on the other hand, is primarily related to ATMs. Topic 3 comprises matters pertaining to loans, staff and consultants, along with which reference is typically also made to the customer or client. Due to its association with primarily numerical characters, the rating indices 1 and 2 that were excluded from the numerical grouping step during preprocessing and the terms *rate* and *sorry*, Topic 4 was deemed to relate to cases in which customers misunderstood the rating scale and were, in fact, correcting this misunderstanding. Finally, Topic 5 is related to the bank's service and debit orders, which appear to be described frequently by adjectives such as *good*, *great*, and *bad*, and the adverb *very*.

Topic	Frequent *salient* keywords	*Relevant* keywords
1	money, get, good, rate, not	no, money, account, bank, get, people, good
2	ATM, money, help, no, want, long, ok	no, not, ATM, bank, help, _bankname, want, money
3	loan, help, client, get, staff, customer, consultant, give	loan, not, get, need, client, consultant, customer, staff
4	_num, 1, sorry, number, rate, long	_num, loan, card, 1, 2, number, sorry
5	service, very, bad, good, great, debit	service, bank, very, _bankname, bad, good, great, debit

TABLE 9.4: *The keywords associated with each topic in Figure 9.34 based on both the frequency with which salient words were observed in and the relevance of words to a given topic.*

In order to determine which of these concerns are the most pressing for the customers, the number of reviews in each sentiment class that contain the keywords or noun phrases identified from the LDA topic model and the word cloud in Figure 9.33(d) were compared by means of a count plot. The resulting plot is shown in Figure 9.35. From this graph, it is clear that *money, loan, help, ATM* and *service* are the most prominent keywords from the five extracted topics, whilst the keyword *time* (which does not feature in the topic analysis) is also relatively prominent. It is interesting to note here that *service* is the only keyword mentioned in almost as many positive reviews as negative reviews.

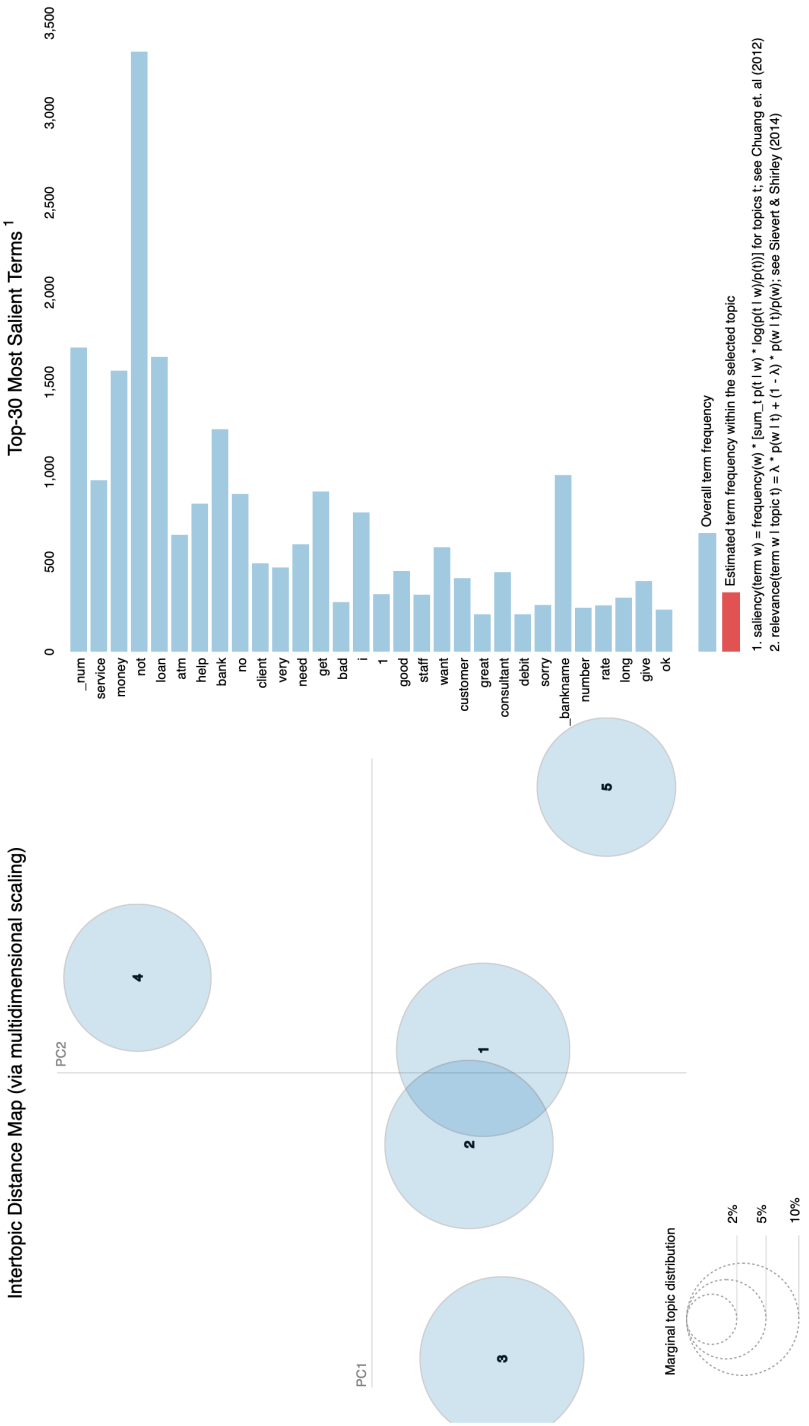


FIGURE 9.34: *The results of the LDA topic model based on five topics and two passes through the case study data.*

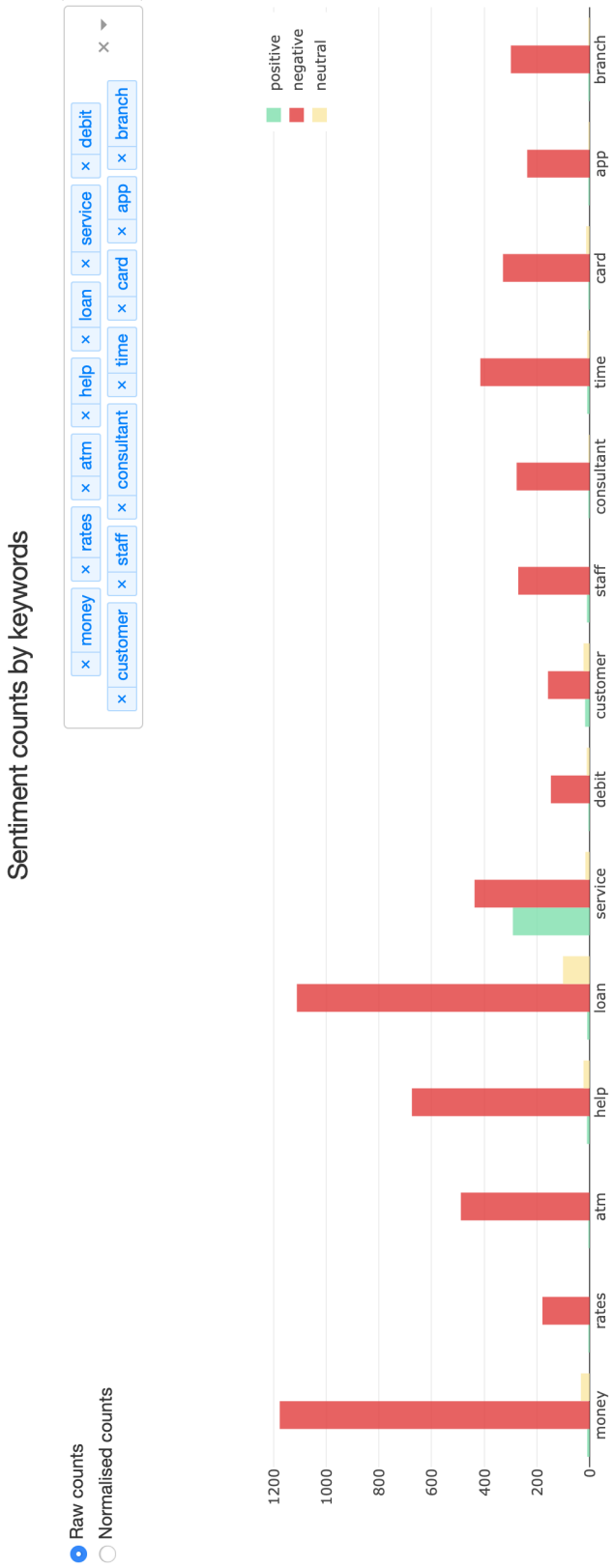


FIGURE 9.35: *The frequency with which selected keywords were observed in the corpus.*

The filtering function of the ECCO system was subsequently employed to gain further insight into what, in particular, customers may have been dissatisfied with in relation to the above-mentioned *prominent* keywords (those that were observed in more than 400 negative reviews), with the addition of the keywords *staff* and *consultant*, which relate to the same concept and have a combined observed frequency greater than 400. To this end, the word clouds of the negative reviews in each filtered selection were scrutinised for informative words and phrases related to the keyword. Furthermore, a few random samples of original reviews were drawn from each selection in order to contextualise these phrases.

The word cloud of negative reviews mentioning the keyword *loan*, for example, is shown in Figure 9.36. Among the most frequently used phrases are *get*, *want*, *need*, *help* and *give*, as well as *qualify*, *declined*, *refused* and *apply*. This suggests that many customers are concerned that they did not qualify for or receive a loan. The following extracted sample reviews confirmed this intuition:

(i) "*They dont want to gave me loan even though they termanite my credit facility*,"

(ii) "*Poor service's....long queues and also high loan interests*,"

(iii) "*Telling me I can take a loan and after saying system does not agree*,"

(iv) "*U couldn't help me with a loan, so I've decided I'm gonna change banks*,"

(v) "*They refuse to borrow me money with R3000 remaing balance on my current loan i was planning to top up it and they didnt explained me why they refuse*," and

(vi) "*I bank with you back i can not get malti loan from you. Now I get loan from [another bank] but I bank with _bankName*."

Variations of the term *pay* are also frequently used in these reviews, possibly indicating some issues related to loan repayments.

The word clouds and sample reviews for the remaining keywords can be found in Appendix B. From these analyses, it was found that the keyword *money* was used in a variety of contexts, but that customers often mentioned having trouble *reversing* money, *drawing* money at ATMs or *borrowing* money in the form of personal loans. Many customers also complained about *needing* money, although this provides little actionable insight for the bank.

The word *help* typically appeared in negative reviews to refer to a lack of assistance received from staff members, especially in acquiring a loan, and to complain about unfriendly service. The related keyword *staff* was similarly associated with complaints of unfriendly service from poorly trained employees, as well as too few staff members. These sentiments were shared in reviews mentioning the keyword *consultant*, in which staff members were described as unprofessional, uninformed and rude with several complaints mentioning that consultants were preoccupied with personal conversations. Finally, reviews which contain the word *service* reflected similar issues. Interestingly, many of the sampled negative reviews in this selection described having had both positive and negative experiences related to customer service. This impression is fortified by the fact that, although only ratings of 2 and 3 were followed up on during the customer survey, almost half of the reviews that mentioned service were, in fact, positive in sentiment, describing the service as *great* or *excellent*. It may thus be concluded that customer service in the bank is inconsistent.

With regards to *ATM*s, customers were found to complain primarily about the insufficient number of available ATMs, especially those with *depositing* capabilities, as well as about the

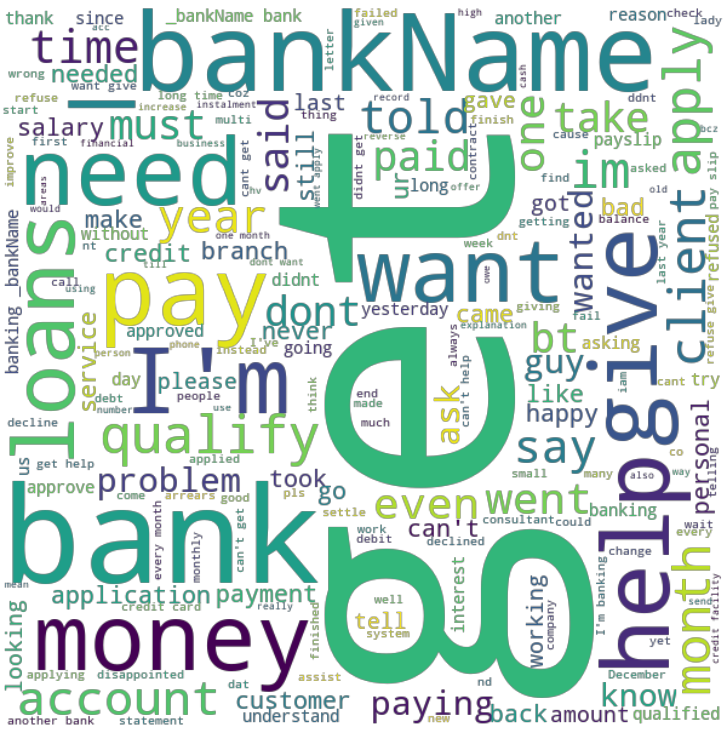


FIGURE 9.36: *Word cloud representation of negative reviews containing the keyword loan (with the keyword itself excluded from the word cloud).*

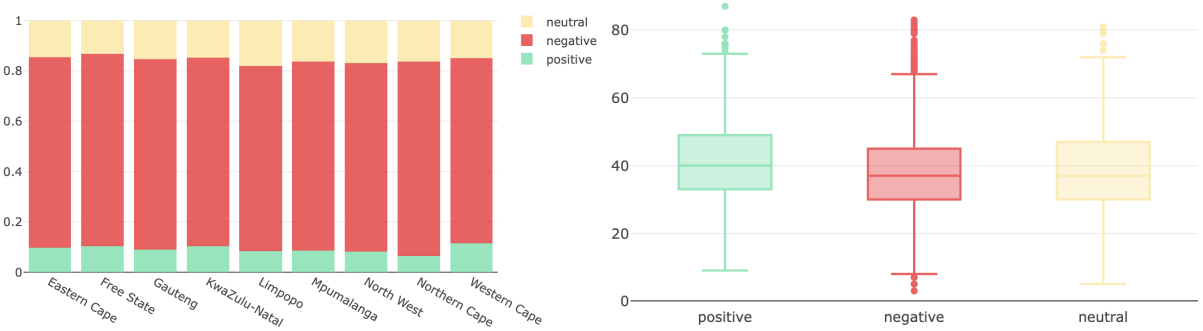
associated long queues and waiting times. Furthermore, customers lamented that machines were frequently *slow*, *faulty* or *offline*. Finally, several customers requested additional security measures in the vicinity of ATMs due to frequent cases of robbery.

The final *prominent* keyword investigated was *time*. It became clear during the analysis, however, that the high frequency of this term was not due to its use as some indicator of the bank's performance but rather as a generic term in several different contexts (*e.g. "every time," "place and time," "the machine timed out,"* and "*the staff took their time.*") This keyword was therefore not further investigated.

Having successfully established the contents of the customer complaints by means of the *summary view* and *topic analysis* tabs of the ECCO system, as well as its filtering function, the next step in using the ECCO framework was an investigation of the relationship between the distribution of sentiment polarity and the structured supplementary data. In this particular case study, however, the value of such an analysis is questionable.

This is due to the fact that many of the positive reviews in the corpus represent cases where customers misunderstood the rating scale or accidentally submitted the wrong rating. It is therefore unclear whether any identified relationships between structured variables and sentiment polarity describe customer types that are more likely to be confused by the rating scale or customers who are more likely to be satisfied with their experience at the bank. In fact, for most of the examined variables, the distribution of sentiment polarity was approximately equal across all possible categories, as shown, for example, in Figure 9.37.

One variable for which a difference in sentiment class distribution was exhibited is the *Q01 Value*, describes the numerical rating submitted by the customer prior to the unstructured review. As shown in Figure 9.38(a), the proportion of reviews classified as *negative* in sentiment is higher for those reviews which followed a rating of 3 (85%) than for those which followed an



(a) *Normalised sentiment counts by province* (b) *Customer age distribution by sentiment class*

FIGURE 9.37: *Examples illustrating the approximately equal distribution of sentiment classes for various values of structured data attributes.*

initial rating of 2 (60%). An alternative interpretation of the graph is that most customers who submitted a free-form response bearing negative sentiment had submitted a rating of 3, whilst customers whose unstructured review was classified as *neutral* or *positive* were more likely to have submitted a rating of 2. This second interpretation offers an approach by which insight can still be drawn from structured variables, even when the sentiment distribution is equal across all categories.

From Figure 9.38(c) and 9.38(d), for example, it is clear that most customers who complained of loans or mentioned the associated keyword *help* rated the bank with a 3. These keywords, being among the three most frequently mentioned keywords, therefore also appear to be a considerable source of dissatisfaction for customers. The keyword *ATM*, on the other hand, while frequently mentioned by customers, did not appear to result in as negative an overall rating. In fact, as shown in Figure 9.38(b), most customers who mentioned ATMs in their subsequent reviews evaluated the bank with a rating of 2 out of 3. The keyword *service* once again represents a mixed case, as shown in Figure 9.38(f). As expected, respondents whose reviews were positive in sentiment typically assigned a rating score of 2, whilst respondents whose reviews were classified as *negative* were more likely to have submitted a rating score of 3. It therefore appears that customers who were dissatisfied with the service were greatly dissatisfied with the experience as a whole, whilst many of the customers who rated the bank only as a 2 out of 3 were, in fact, satisfied with the service (it is unknown how many of these customers intended to submit positive rating scores). Finally, the keywords *staff* and *consultant* in Figures 9.38(e) and 9.38(g) exhibit similar cases of rating scores 2 and 3, and a majority of rating score 3, respectively. This may indicate that the general staff was less of a concern for customers than the particular consultant who was assisting the customer with their query.

Analysing this attribute for each of the prominent keywords thus revealed that customers who submitted reviews of negative sentiment with respect to certain topics, namely ATMs and staff, were more lenient on their numerical ratings. In this manner, it could be gauged how important certain topics are for overall customer satisfaction. In the remainder of this section, a similar approach is followed to investigate the dominant characteristics of the customers who submitted negative reviews and the branches that they visited prior to submitting these reviews. These results are interpreted, where possible, in comparison with the expected characteristics for the average customer of the bank. Alternatively, differences in dominant characteristics for reviews that mentioned certain keywords are identified.

(a) *All*

(b) *ATM*

(c) *loan*

(d) *help*

(e) *staff*

(f) *service*

(g) *consultant*

FIGURE 9.38: *Sentiment counts by Q01 Value for (a) all reviews and (b)–(g) reviews that contain various keywords.*

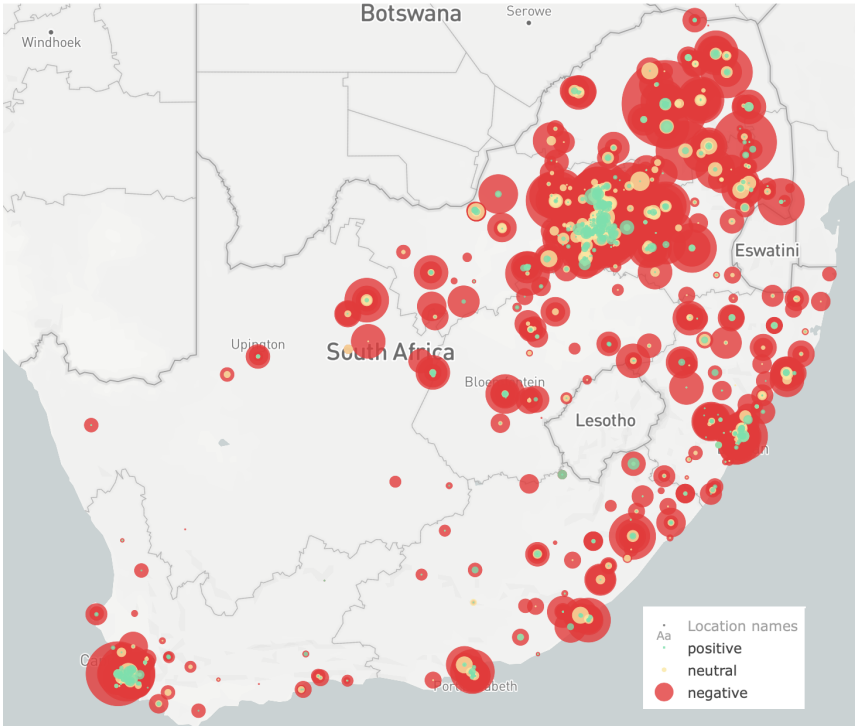
The data attributes related to the branch visit itself are the branch location and type, the primary need of the customer when visiting the branch, the type of consultant that attended to the customer and the date of the branch visit. A bubble map representation of the case study data is shown in Figure 9.39(a). There is a clear dominance of negative sentiment throughout the country with little evidence to suggest that customers in any particular area were more likely to submit positive reviews following their non-positive rating[5]. Most branch locations for which several data points were observed exhibited a majority of negative reviews and a small fraction of positive and neutral reviews. The majority of reviews were submitted by customers situated in Gauteng, particularly near Johannesburg and Pretoria. Furthermore, a large portion of reviews were sent from Cape Town. The remaining reviews originiated primarily from other large cities such as Durban, Port Elizabeth, East London, Bloemfontein and Kimberly, or near the eastern and southern coastline of the country. This distribution is consistent with the population density of South Africa shown in Figure 9.39(b). Since a greater number of reviews can be expected from more densely populated areas, no unusual *hotspots* of dissatisfied customers could be identified by means of this graph.

A similar result was obtained from the count plots visualising the number of reviews in each sentiment category by province, as shown in Figure 9.40(a). From this representation, it is clear that the majority of reviews were associated with branches located in Gauteng, followed by KwaZulu-Natal with less than half the number of associated reviews. Limpopo, Mpumalanga and the Western Cape exhibit an approximately equal number of reviews — slightly less than KwaZulu-Natal. Finally, the smallest numbers of reviews are associated with the Eastern Cape, North-West province, Free State and Northern Cape, in decreasing order.

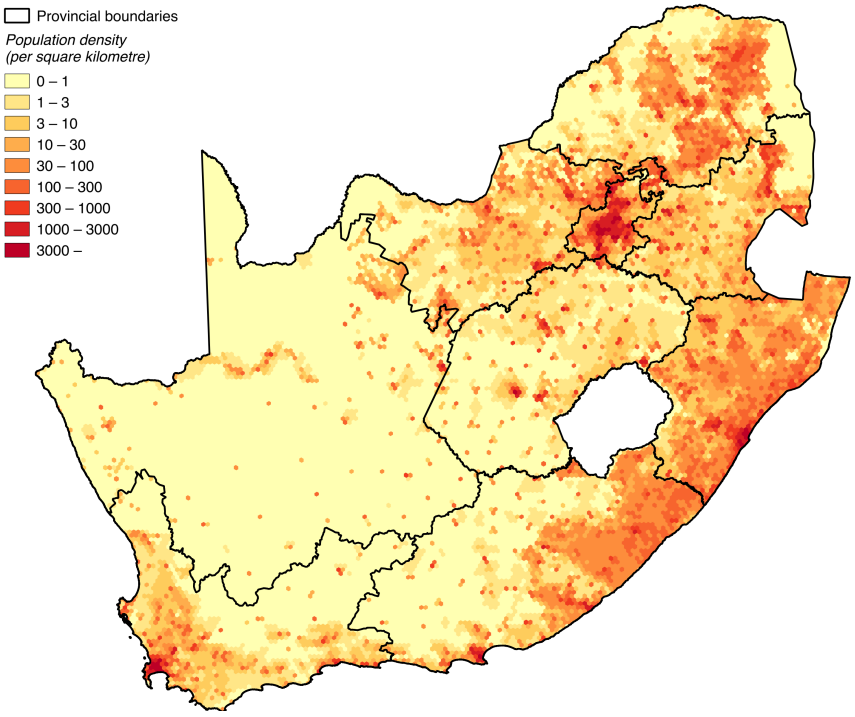
Since not all branch and customer records were available for this case study, it is unclear how this distribution compares with the overall distribution of the industry partner's branch or customer locations. It is, however, interesting that a similar distribution was uncovered for reviews containing the keywords *loan, service, consultant* and *help*, whilst the number of reviews containing the keywords *ATM* and *staff* appear to be distributed differently across provinces. From Figure 9.40(b), it is clear that the Eastern Cape, Free State, KwaZulu-Natal, Limpopo, Mpumalanga and North-West provinces are all more strongly represented by reviews relating to ATMs than in the general case in Figure 9.40(a). It is possible that the bank has expended fewer resources to install and service ATMs in less densely populated areas, resulting in a higher relative level of customer dissatisfaction with respect to this topic in the affected provinces. Reviews which mention the keyword *staff*, on the other hand, exhibit an increase in the relative number of reviews associated with branches in KwaZulu-Natal and a decrease in the number of reviews associated with Limpopo compared with the general case in Figure 9.40(a). This indicates that customers in KwaZulu-Natal are especially dissatisfied with the staff, whilst other issues prove more important in Limpopo.

On a similar note, the distribution of reviews with respect to branch types was investigated. Of all reviews analysed, approximately 50% were associated with urban branches, 20% were associated with branches in rural areas and 30% were associated with branches semi-rural areas, as illustrated in Figure 9.41(a). A similar distribution was exhibited for reviews mentioning the keywords *loan*, *service* and *help*. As shown in Figures 9.41(b) and 9.41(d), however, the proportion of reviews associated with urban branches rises to over 60% and 64% for reviews pertaining to *staff* and *consultants*, respectively. The behaviour of customer service staff thus seems to be a greater cause for concern in metropolitan areas. More interestingly, the proportions of reviews associated with urban, rural and semi-rural branches are 40%, 25% and 35%, respectively, for

[5]Note that what appears to be a culmination of positive sentiment in the north-eastern part of the map merely represents an overlay of various circles with similar centre points, as can be verified by zooming in on the map.



(a) *Bubble map representation of the case study sentiment distribution.*



(b) *Population density of South Africa in 2011 (illustration created by Adrian Frith [90] from census data by Stats SA [298]).*

FIGURE 9.39: *A comparison of (a) the bubble map of the case study data representing the distribution of sentiment over South Africa and (b) the population density of South Africa.*

(a) *All reviews*

(b) *Keyword ATM*

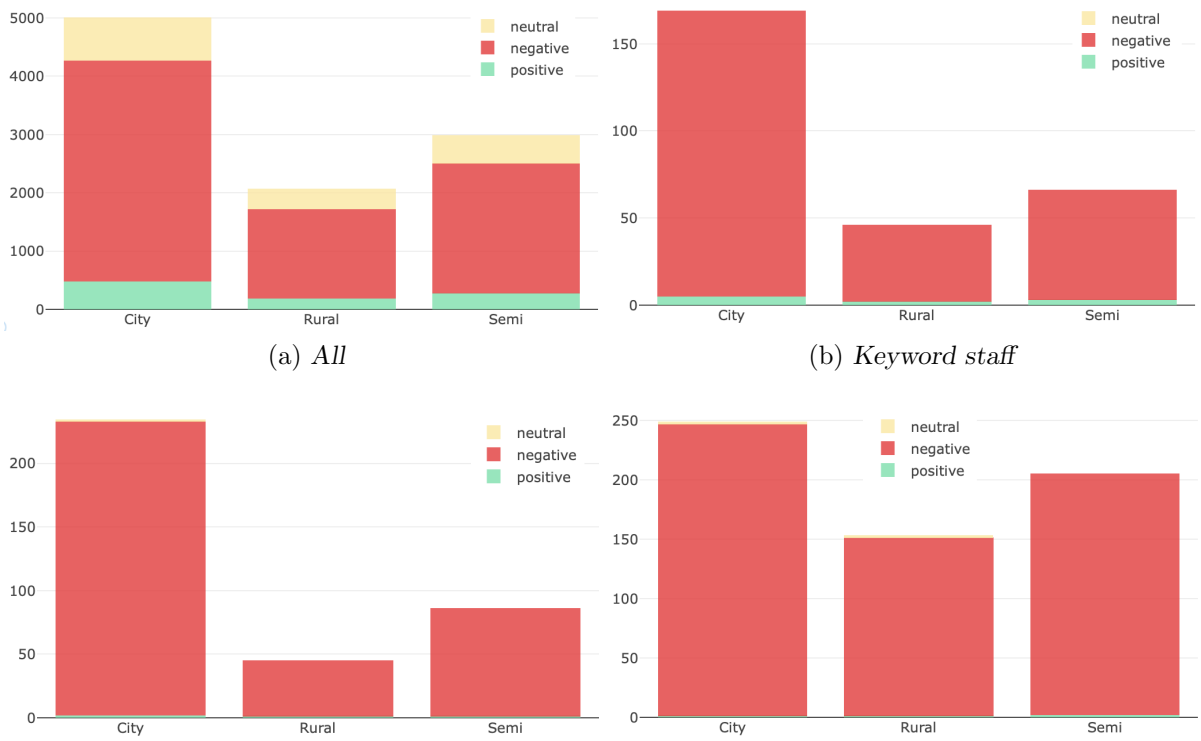
(c) *Keyword staff*

FIGURE 9.40: *Sentiment counts by branch province for (a) all reviews, (b) reviews that contain the keyword ATM and (c) reviews that contain the keyword staff.*

reviews that mention ATMs. This confirms the theory that ATM machines pose a significant problem, particularly in rural and semi-rural areas.

With respect to the primary need of the customer, Figure 9.42 reveals that most dissatisfied customers were attempting to complete a transaction during their branch visit. Several customers also intended to make a general enquiry or apply for credit, whilst fewer customers visited the branch for matters related to savings, deposits and the opening of new accounts, as shown in Figure 9.42(a). As is evident from Figure 9.42(b), customers who complained about staff members were more likely to require assistance with enquiries and savings. Furthermore, as expected, customers who mentioned loans in subsequent reviews of their experience were more likely to have visited the branch to discuss matters related to credit. It is interesting that the proportion of customer reviews associated with credit matters also increased for reviews mentioning the keyword *help*, which reflects the finding from Table 9.4 that the words *loan* and *help* were typically mentioned in the same reviews.

The consultant type was a *consultant* rather than a cashier in at least 80% of reviews irrespective of the keyword mentioned. Finally, the number of reviews received per day is illustrated in Figure 9.43. As is evident from Figure 9.43(a), the case study data span a period of just over a year from February 2017 to March 2018. Up to November 2017, the number of reviews received per day averaged approximately five reviews per day with a slightly lower frequency exhibited between May and September 2017. In December 2017, a rapid increase was observed, leading to a higher average frequency of close to ten reviews per day for the remainder of the period. Generating time series graphs for reviews in which certain keywords were mentioned produced a similar pattern with a distinctly higher frequency from December 2017 onwards and a lower number of observations between May and September 2017. An example of this pattern is shown



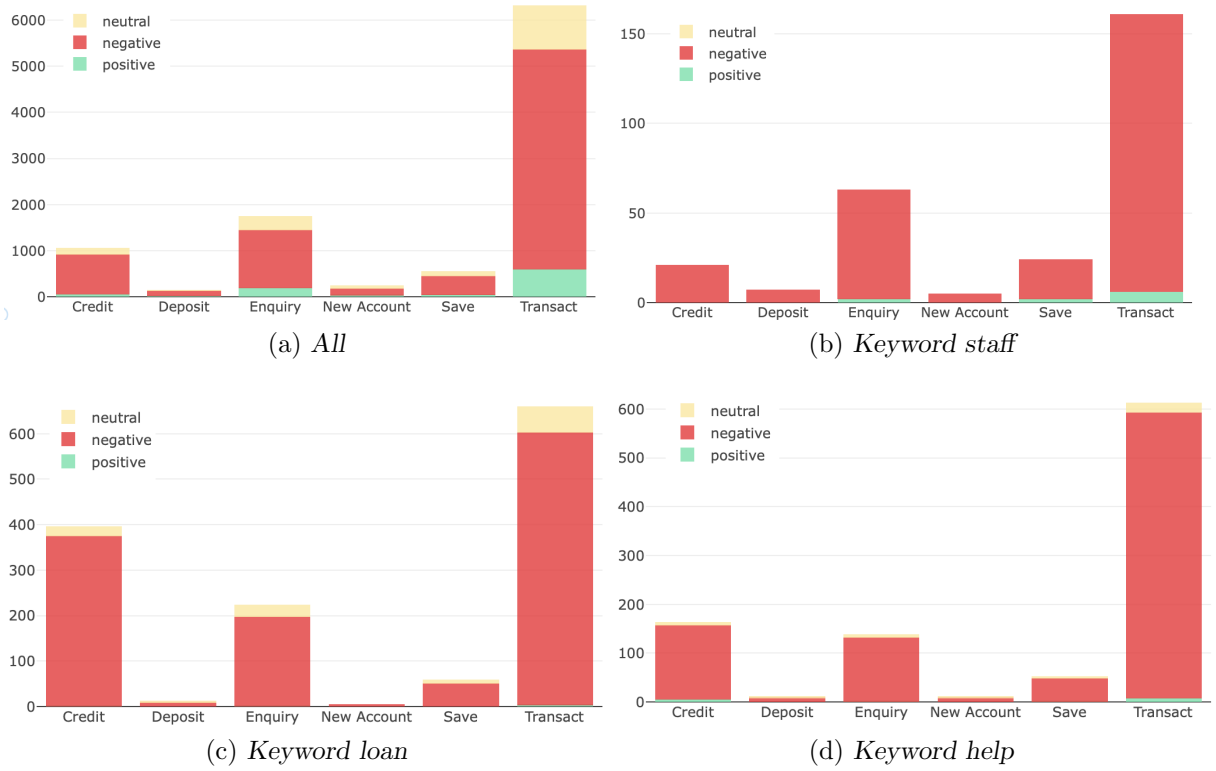
(a) *All* (b) *Keyword staff*

(c) *Keyword consultant* (d) *Keyword ATM*

FIGURE 9.41: *Sentiment counts by branch type for (a) all reviews, (b) reviews that contain the keyword staff, (c) reviews that contain the keyword consultant and (d) reviews that contain the keyword ATM.*

for the keyword *consultant* in Figure 9.43(b). The sharp peak at the end of November 2017 shown in Figure 9.43(a) is, however, only exhibited in the graph for the keyword *loan*, as shown in Figure 9.43(a). Whilst there are a number of possible reasons for the general increase in negative reviews, customer complaints about loans were distinctly more frequent towards the end of the year, possibly indicating that a larger number of customers required loans at this time due to planned expenses during the festive season. The day of the week on which reviews were received did not appear to have a significant effect, with an approximately equal number of reviews received on each work day, and few on a Saturday, in line with expectations.

The attributes describing the customers themselves include demographic information such as their town and country of residence, gender, age and marital status. Furthermore, information more relevant to the bank included the customers' primary bank, service plan with the industry partner, average monthly fee, internet banking usage and registration status, as well as their loan and transaction status. Finally, information related to the clients' salary and cellular network was also available. Many of these attributes did not reveal particularly interesting insights, since, as previously mentioned, the attributes of the bank's average customer were not available for comparison. Furthermore, a meaningful analysis was not possible in some cases, either because there were too many categories for an attribute, resulting in too small a sample size for each category (*e.g.* the customer's city of residence), or because most observations fell into a single category for an attribute (*e.g.* the customer's country of residence). Some attributes, however, revealed interesting patterns when compared for the subsets of reviews that mention certain keywords.



(a) *All* (b) *Keyword staff*

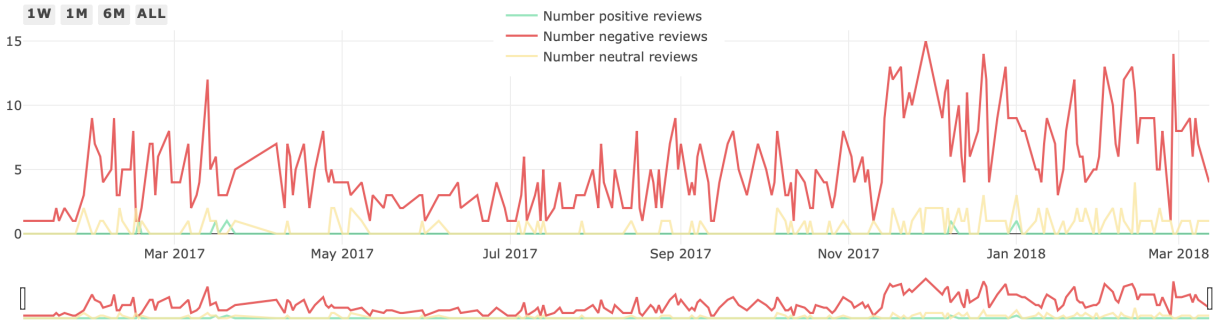
(c) *Keyword loan* (d) *Keyword help*

FIGURE 9.42: *Sentiment counts by primary need for (a) all reviews, (b) reviews that contain the keyword staff, (c) reviews that contain the keyword loan and (d) reviews that contain the keyword help.*

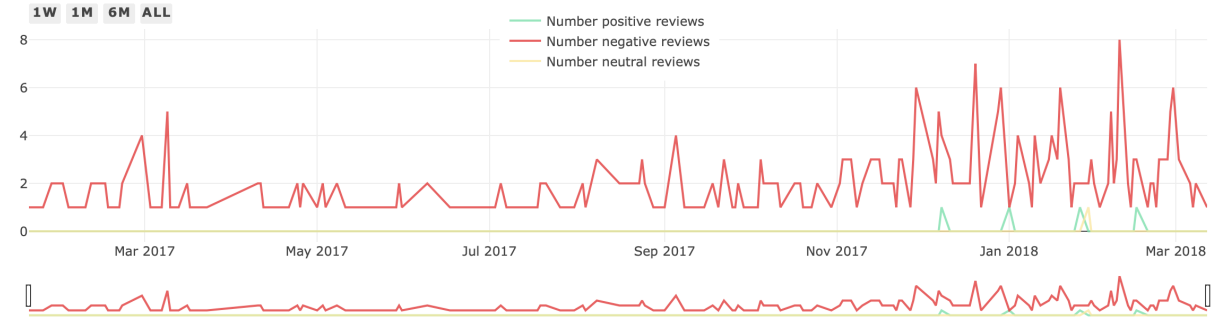
The median of the average monthly fee paid by customers who mentioned ATM in their reviews, for example, was R87, compared with the R70–R75 median fees paid monthly by customers complaining of other matters. These customers, who complain primarily of a lack of ATMs, thus incurred an increase in fees of over 15% — likely because of surcharges incurred for drawing money at the machines of other banks.

Furthermore, the median salary of all surveyed customers was R6 475 and varied between R6 368 and R6 774 for the keywords *ATM*, *loan*, *help* and *service*. The median salary earned by customers complaining about *staff* and *consultants*, on the other hand, was R7 909 and R9 000, respectively. Higher earning customers therefore seemed to be more concerned with the attitude and aptitude of customer contact employees than customers who earn less.

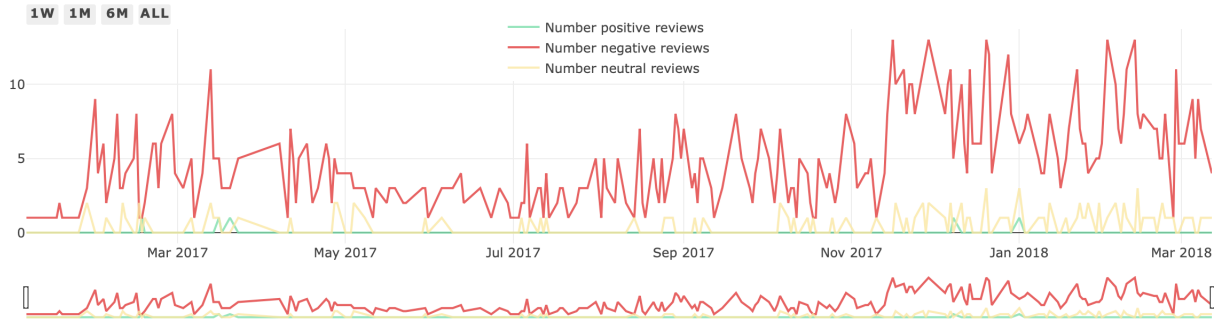
Finally, the attribute describing a customer's loan status revealed a similar pattern to the general case shown in Figure 9.44(a) for most keywords. As is evident from the figure, most clients (43%) who reviewed the bank did not have a loan at the time of the survey. In contrast, 19% of customers did have active loans, whilst 9% were in arrears with their loan repayments and a further 9% and 3% of customers had loans that were dormant or in some form inactive, respectively. Focusing exclusively on customers who submitted reviews that mention the keyword *loan* produced a different distribution. As shown in Figure 9.44(b), the majority of these customers (33%) had loans that were classified as dormant, whilst a further 26% had active loans, 6% had inactive loans and 12% were falling short on loan repayments. The proportion of customers without loans was 22% in this case. Most of the customers who complained about loans thus appear to be those customers who had, in some form or another, not been able to keep up to date with their existing loan repayments. This information, in conjunction with the content analysis



(a) *All*



(b) *Keyword consultant*



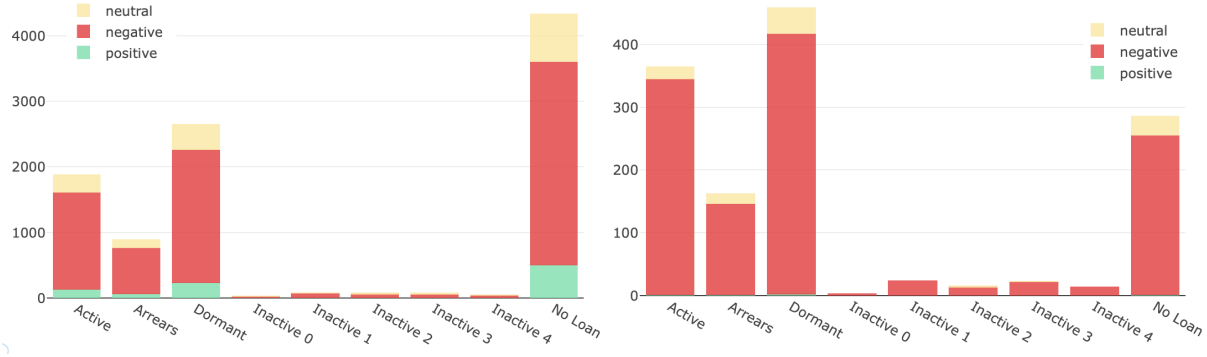
(c) *Keyword loan*

FIGURE 9.43: *Sentiment counts by date for (a) all reviews, (b) reviews that contain the keyword consultant and (c) reviews that contain the keyword loan.*

that was previously performed in respect of this subset of reviews, leads to the conclusion that many customers are dissatisfied that they could not acquire a *second* loan from the bank.

The final step in the ECCO framework is to perform a multivariate analysis in order to identify latent relationships between structured variables and the sentiment class. Due to the high class imbalance of the data and the difficulty of interpreting such relationships, given the possibility of accidental incorrect ratings by customers as described earlier, this analysis was not performed for the general case study data. Since the subset of reviews which mention the keyword *service* exhibited an approximately equal distribution of positive and negative sentiment, however, a decision tree could be fit to this subset of the data.

After some experimentation with the available hyperparameters, a decision tree was fit by means of the Gini impurity index subject to the constraints that 15% of the observations were required to fall into a particular category to warrant a split on the associated feature and that at least



(a) *All* (b) *Keyword loan*

FIGURE 9.44: *Sentiment counts by loan status for (a) all reviews and (b) reviews that contain the keyword loan.*

20% of observations were represented in a given leaf node. The resulting tree, which achieved a validation accuracy of 72.5%, is shown in Figure 9.45.

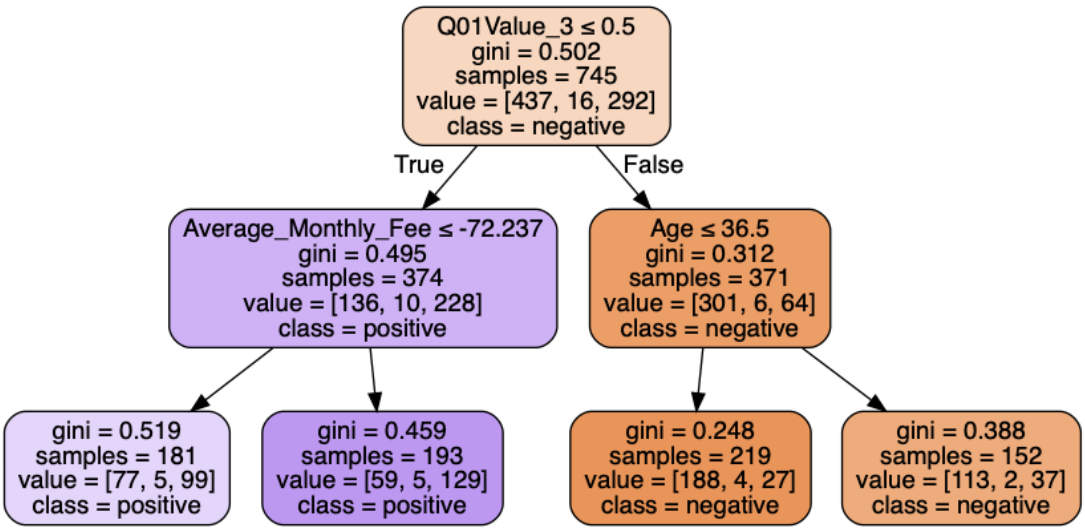


FIGURE 9.45: *A decision tree fit to the subset of reviews that mention the keyword service, using the Gini impurity index subject to a minimum proportion of 15% of observations to warrant the split of a node and a minimum proportion of 20% of observations to be present in any given leaf node.*

The top split of the decision tree is the Q01 value, which was already shown to distinguish effectively between positive and negative observations in Figure 9.38(f). The left-hand branch of the root of the tree (observations for which the initial rating was not equal to 3) is classified as positive, whereas the right-hand branch of the tree (observations for which the initial rating was not equal to 3) is classified as negative. The decision tree further splits on the attributes describing the average monthly fee paid by customers and the customers' age for the positive and negative branches, respectively. More specifically, the model is *'more certain'* that a given customer submitted a positive review after giving the bank a rating score of 2 out of 3 if the customer's average monthly banking fee is larger than −R72.237 (*i.e.* less than R72.237 per month). Furthermore, the model is *'more certain'* that a review following a rating of 3 is negative in sentiment if the customer is 36 years old or younger.

A similar result was achieved when a decision tree was fit by means of the information gain criterion and a minimum of 20% of observations required in leaf nodes as well as to warrant a split. The resulting tree, which achieved a validation accuracy of 73.8%, is shown in Figure 9.46. As can be seen in the figure, the top split and second split on the right-hand branch of the tree is the same as in Figure 9.45. The second split on the left-hand branch of the tree, however, is the same as for the right-hand branch. In this case, the model is more confident in its prediction of the positive class when customers are 37 years or older.

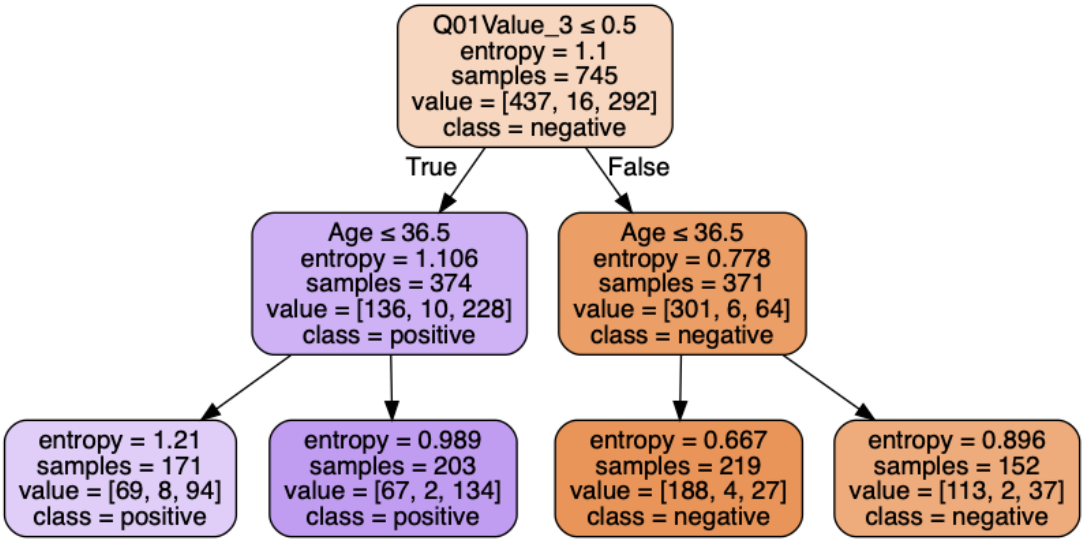


FIGURE 9.46: *A decision tree fit to the subset of reviews that mention the keyword service, using the information gain splitting criterion subject to a minimum proportion of 20% of observations to warrant the split of a node and a minimum proportion of 20% of observations to be present in any given leaf node.*

Finally, the same validation accuracy was achieved when the required observations in a leaf node is increased to 25%. The decision tree produced in this case is shown in Figure 9.47. The right-hand branch of the tree is not split further for this set of hyperparameters and the left-hand branch is split on the attribute *salary*. More specifically, the model is more confident in its prediction of the positive class for customers who rated the bank with a 2 and who earn less than R5 306.00 per month. Synthesising the insights conveyed by all of these decision trees, it appears that customers who mentioned the keyword *service* in their reviews were most likely to convey negative sentiment if their initial rating of the bank was 3 and they were 36 years old or younger. Positive sentiments were, on the other hand, more often conveyed by older customers who submitted an initial rating of 2 and were on the lower end of the spectrum with respect to both their average monthly banking fee and their salary.

9.4 Chapter summary

In this chapter, the results of the case study analysis were presented. The processing steps performed on the data were first described, illustrating to what extent each of the filtering and normalisation steps contributed to the reduction of the vocabulary size, resulting in a final vocabulary that was over 60% smaller than the original, unprocessed vocabulary. Subsequently, the model development process was delineated, including the approach taken to tune various hyperparameters and the evaluation of the models generated by the ECCO system as well as the model employed by the third-party vendor contracted by the industry partner. Finally, the results of the selected CNN model were analysed by means of the *Dashboard* interface, revealing

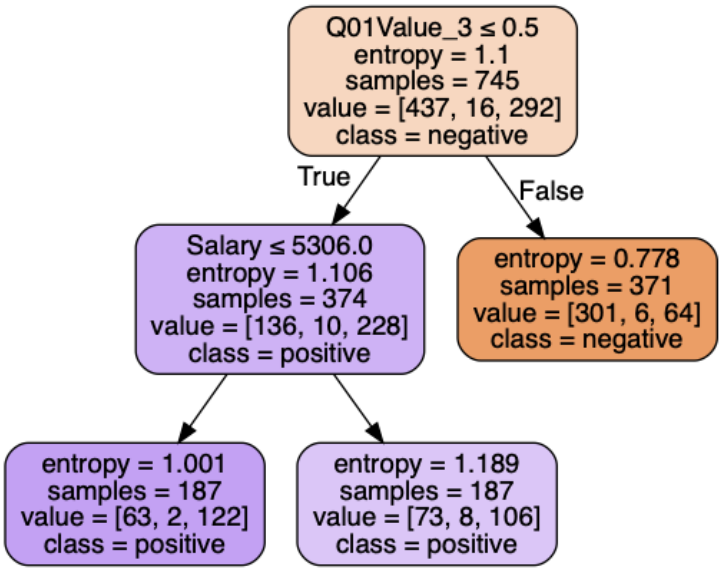


FIGURE 9.47: *A decision tree fit to the subset of reviews that mention the keyword service, using the information gain splitting criterion subject to a minimum proportion of 20% of observations to warrant the split of a node and a minimum proportion of 25% of observations to be present in any given leaf node.*

the contents of the customer reviews, giving insight into the importance of various topics to customers and attempting to find associated patterns in branch and customer attributes. The findings of the case study analysis may be summarised by answering the questions posed in §8.3 as follows.

Research question (i): *Is it possible to develop a model that outperforms the sentiment analysis software of the third-party vendor using the ECCO system?*

> The accuracy and AUC score achieved in respect of the case study data by the software of the third-party vendor are 59.05% and 0.6602, respectively. The machine learning models developed by means of the ECCO system, on the other hand, achieved median accuracies between 82% and 85%, and median AUC scores between 0.79 and 0.91 for ten replications, with four of the five models achieving AUC scores above 0.89. It is, therefore, possible to develop a model that outperforms that of the third-party vendor by a large margin.

Research question (ii): *Which types of models perform better in this context?*

> The lexicon-based models evaluated during this case study achieved median accuracies in the range of 39%–48% and median AUC scores between 0.60–0.62 for the ten test data sets. The results of the machine learning models mentioned in (i) therefore prove superior by a large margin. The performance of the five tested machine learning algorithms was competitive for both metrics, with the naïve Bayes model performing slightly worse than the other four models in terms of accuracy and the SVM model performing slightly worse in respect of the AUC score (which may have been caused by an inaccurate estimate of the score for this particular model). Furthermore, machine learning models employing various document representations achieved competitive performance. Models employing feature vectors representing unigrams or unigrams in combination with bigrams, moreover, performed significantly better in respect of both accuracy and AUC scores than those models which employed feature vectors representing bigrams only.

Ensemble models were, furthermore, shown to improve performance over their individual components by up to 2%. This effect was observed for both for ensembles comprising only the more accurate machine learning models and for ensembles employing both machine learning and lexicon-based models as base learners. The most consistent performance was achieved by combining scoring outputs of base learners by means of a meta-learning approach (or *stacking*). Furthermore, in terms of base learner selection, a new favourability metric quantifying the trade-off between ensemble diversity, accuracy and size showed promising results compared to a simple greedy approach focused on accuracy. The benefit of the proposed approach is that ensemble diversity may be computed prior to deploying individual models and without storing detailed classification results.

Research question (iii): *How many of the reviews received from customers did, in fact, contain negative comments?*

According to the chosen CNN model that was deployed on the case study data, 75.10% of the total reviews conveyed negative sentiment, whilst 9.39% conveyed positive sentiment and 15.5% bore no sentiment (were classified as *neutral*). The software of the third-party vendor, on the other hand, classified 55.79% of its input data as *negative*, 2.65% as *positive* and 41.55% as *neutral*.

Research question (iv): *What are the reasons for the dissatisfaction of customers in these cases?*

The most prominent points of concern for customers were matters related to personal *loans* and a lack of *help* offered by assistants in acquiring these loans and understanding, where applicable, why applications were rejected. Furthermore, many customers complained of a lack of *ATMs*, as well as a frequent occurrence of faulty or broken machines. *Service* appears to be inconsistent from the perspective of customers, with widely varying evaluations of service quality, even in individual reviews. Finally, several customers also complained of unfriendly, unprofessional and poorly trained *staff* or *consultants*. Judging by the overall rating given by customers who mentioned each of these keywords, matters related to *loans* and *help* received from individual consultants appeared to influence the overall rating of the bank more severely than matters related to *ATMs* or the general customer contact *staff*. Matters related to *service*, once again, produced varying results.

Research question (v): *Are there any other distinguishable trends indicating that certain branches or customer profiles are more dissatisfied than others?*

From a comparison of the distribution of structured variables for various subsets of the data, complaints about a lack of *ATMs* appear to be of greater concern in rural and semi-rural areas, especially in the Eastern Cape, Free State, KwaZulu-Natal, Limpopo, Mpumalanga and North-West provinces. Furthermore, these complaints appear to come from customers incurring, on average, over 15% higher monthly fees than other customers. Matters related to *staff* and *consultants*, on the other hand, seem to be of greater concern to customers in urban areas, particularly those located in KwaZulu-Natal. These customers, furthermore earn considerably more than the average customer in terms of a monthly salary. Negative reviews related to *loans* appeared to originate primarily from customers attempting to apply for a secondary loan, often having fallen into arrears or into a dormant state in respect of their current loan. Moreover, such complaints were received in significantly greater volumes towards the festive season at the end of the year. Finally, reviews related to the bank's general *service* tended to produce negative results for younger, higher-earning customers compared with those customers who submitted positive reviews on this matter.

Part IV

Framework validation

CHAPTER 10

Benchmark data sets

Contents

In the previous part of this dissertation, the ECCO system was employed to conduct a case study in the banking domain, illustrating how the ECCO framework may be applied in a real-world scenario and showcasing its ability to facilitate the extraction of actionable insight from raw, opinon-bearing data. In order to further validate the ECCO framework and to illustrate its *general* applicability, additional case studies were conducted in respect of data from various domains. Three additional data sets were selected from the literature for this purpose. These data have been employed by other researchers and may, therefore, also serve as benchmark for the sentiment modelling component of the framework. In this chapter, each of these benchmark data sets is described in turn and any data preparation activities performed in respect of the data are delineated. Furthermore, a short discussion is included on the suitability of this collection of data sets to validate the framework. Validation analyses conducted using these data sets are described in the following chapter.

10.1 The PL04 data set

The *PL04* data set[1] contains 1 000 *positive* and 1 000 *negative* processed movie reviews, all written before 2002. The data set was originally published by Pang and Lee [230] and has been used as a benchmark data set in many subsequent studies. In order to generate the data set, the authors selected only those reviews which contained some rating information (*e.g. "4 out of 5 stars"*). Ratings of at least 3.5 out of 5 stars, 3 out of 4 stars or letter grade B were considered *positive* whilst ratings of at most 2 out of 5 stars, 1.5 out of 4 stars or letter grade $C-$ were considered *negative*. Such rating information was subsequently removed from the review. Furthermore, a limit of twenty reviews per author per category was imposed.

The data published by Pang and Lee were presented in various formats, including individual text files for each review, separated into two folders labelled *"pos"* and *"neg."* These text files

[1] Available at http://www.cs.cornell.edu/people/pabo/movie-review-data/ as *"polarity data set v2.0."*

were compiled into a single column in a relational data table, and the sentiment classes *positive* and *negative* were entered into the second column according to the folder in which each text file was found. An excerpt of the data set in the format employed in this dissertation is shown in Table 10.1. It is evident from this sample that the reviews had already been processed by Pang and Lee [230]. The text has, for example, been normalised and some form of tokenisation has been applied, often separating punctuation marks from the accompanying words.

Review	Sentiment
this film is extraordinarily horrendous and i'm not going to waste any more words on it .	negative
beware of movies with the director's name in the title . take " john carpenter's ghosts of mars " (please) . if the carpenter brand name wasn't superglued to the title , this embarrassment would surely have bypassed theaters entirely and gone straight to its proper home on the usa network . and i would have been spared a headache . the latest from the director of " starman , " " halloween " and " escape from new york " is a lousy western all gussied up to look like a futuristic horror flick . the production is set on mars in 2176 , where humanity looks for relief from the overpopulation strangling their home world . six hundred and forty thousand people in a matriarchal society live and work at outposts all over the red planet , terra-forming to make it more hospitable for future generations . a matriarchal society . terra-forming . sounds pretty intriguing , eh ? well , don't get your hopes up .	negative
kolya is one of the richest films i've seen in some time . zdenek sverak plays a confirmed old bachelor (who's likely to remain so) , who finds his life as a czech cellist increasingly impacted by the five-year old boy that he's taking care of . though it ends rather abruptly– and i'm whining , 'cause i wanted to spend more time with these characters– the acting , writing , and production values are as high as , if not higher than , comparable american dramas . this father-and-son delight– sverak also wrote the script , while his son , jan , directed– won a golden globe for best foreign language film and , a couple days after i saw it , walked away an oscar . in czech and russian , with english subtitles .	positive
i had been expecting more of this movie than the less than thrilling twister . twister was good but had no real plot and no one to simpithize with . but twister had amazing effects and i was hoping so would volcano volcano starts with tommy lee jones at emo . he worrys about a small earthquake enough to leave his daughter at home with a baby sitter . there is one small quake then another quake . then a geologist points out to tommy that its takes a geologic event to heat millions of gallons of water in 12 hours . a few hours later large amount of ash start to fall . then . . . . it starts . the volcanic eruption . . . . i liked this movie . . . but it was not as great as i hoped . i was still good none the less . it had excellent special effects . the best view . . . the helecopters flying over the streets of volcanos . also . . . there were interesting side stories that made the plot more interesting . so . . . it was good ! !	positive

TABLE 10.1: *An excerpt of the PL04 data set (sic).*

10.2 The Yelp data set

The *Yelp Open Data Set*[2] contains data related to business reviews from the popular review site *Yelp*. These data were presented in several data sets in JSON format, containing reviews submitted on the site, selected attributes of reviewed businesses and site users, tips written by users on selected businesses (*e.g.* "ask for the *chicken special* not listed on the menu"), photographs taken at establishments and time stamps of *check-ins* logged at businesses. For the validation study, the first three of these data sets were selected, namely *review*, *business* and *user*. The attributes contained in each of these data sets are described in Table 10.2.

Since the data were presented in JSON format, each attribute had a varying number of values for each entry. One business may, for example, have three items listed under *category*, whilst another may have none. These items may also be nested. *Attributes* of a business, may for example take the form

"attributes": {
"RestaurantsTakeOut": true,
"BusinessParking": {
"garage": false,
"street": true,
"validated": false,
"lot": false,
"valet": false
},
}.

In order to create relational data tables to employ in the ECCO system, several attributes were extracted from this format. For the *business* data set, the attributes *name, city, state, postal code, latitude, longitude, stars* (the average star rating of the business), *review_count* (the number of reviews written about the business) and *is_open* (a binary variable equal to 1 if the business is currently open) could be employed directly, since only one value was given for each establishment in either numerical or categorical format. Furthermore, the top three categories were extracted from the *categories* attribute in order of the categories' overall frequency in the data set, yielding three additional variables, *top_category_1, top_category_2* and *top_category_3*. Where fewer than three categories were found, "*none*" was entered as the respective table entry. The remaining attributes were discarded due to a lack of consistency across entries.

For the *review* data set, all attributes were retained in their original format, since a single value was provided in each case. The variables *useful*, *funny* and *cool*, represent the number of users who indicated that the review may be associated with the specified property. The perceived usefulness of a review is often an important metric for determining the order in which reviews are presented to users of a review site, and for identifying false reviews. In order to transform the data set into one suitable for classification, a categorical target variable *sentiment* was generated from the star rating given by the reviewer. More specifically, star ratings of 1 or 2 were classified as *negative*, ratings of 4 or 5 were classified as *positive*, and ratings of 3 were discarded, following the same process as that adopted by Salinca [267].

Finally, for the *user* data set, most attributes were retained in their original form. Many of these were similar to attributes already described in respect of the previous data sets, such as *review_count* and *useful*, whilst others may be interpreted in a similar manner (*compliment_hot*,

[2] Available at https://www.yelp.com/dataset.

Data set name	Description	Records	Attributes
Review	Reviews submitted on the site	6 685 900	*review_id* *user_id* *business_id* *stars* *useful* *funny* *cool* *text* *date*
Business	Details of reviewed establishments	192 609	*business_id* *name* *address* *city* *state* *postal_code* *latitude* *longitude* *stars* *review_count* *is_open* *attributes* *categories* *hours*
User	Details of reviewers	1 637 138	*user_id* *name* *review_count* *yelping_since* *useful* *funny* *cool* *elite* *friends* *fans* *average_stars* *compliment_hot* *compliment_more* *compliment_profile* *compliment_cute* *compliment_list* *compliment_note* *compliment_plain* *compliment_cool* *compliment_funny* *compliment_writer* *compliment_photos*

TABLE 10.2: *The individual Yelp data sets employed in the study.*

for example, represents the number of "hot compliments" received by the user). The year in which users signed up to the review site was extracted from the attribute *yelping_since* since this was deemed more informative than the date itself. Furthermore, the attributes *friends* and *fans*, describing associations between users, were disregarded.

The three resulting relational data tables are related to one another in the manner illustrated in the entity relationship diagram in Figure 10.1. More specifically, each review is associated with one user and one business. In accordance with the required input to the ECCO system, two relational data tables were formed. In particular, the *user* data set was chosen as the *supplementary data* set and the *business* and *review* data sets were merged to form the *reviews* data set.

FIGURE 10.1: *An entity relationship diagram of the Yelp data.*

Finally, due to limited computing power[3], the size of the resulting reviews set was reduced by restricting the time frame to the year 2018 and randomly selecting 10% of the reviews on each of the most reviewed establishments (those which were mentioned more than 1 000 times in the 2018 subset).

An excerpt of the review text and sentiment attributes for the reviews data set is given in Table 10.3. Some of the associated structured variables for each of the review texts in Table 10.3 are listed in Table 10.4

Review text	Sentiment
Neat and clean location. First time at this location ... as usual Starbucks is always good ..	positive
GET THE PRETZELS GUYS. Also, ask for Alex! The service here was amazing. Alex was the best waiter I've ever had. He was tentative to our water needs, constantly making sure we had enough. We also had to wait a little extra long for our pretzels so he GAVE THEM TO US FOR FREE. Not only that, but his smile lit up the space and he was so nice. I felt so welcome! Definitely coming here again! 1000/10. Give this guy a raise!	positive
Drive through window girl was incredibly rude on two separate occasions. Food was ok.	negative
Hmmm let's seeeee go to this McDonald's again or have my teeth ripped out with someone's rusty screw driver? Yeah I'd rather go with the screw driver to be perfectly honest. I have never in my life seen a McDonalds that gets the order wrong EVERY SINGLE TIME. I even had a simple SANDWICH screwed up! JUST ONE SANDWICH!! How?? I understand flipping burgers is hard and reading the order from the screen?! Forget about it! How is someone supposed to make a burger while reading an order?! (Sarcasm). *sigh*... don't even bother with this place. I'd rather just eat dirt.	negative

TABLE 10.3: *An excerpt of the Yelp reviews data set (sic).*

[3]Memory requirements for operations on the original data set, which contained 6 685 900 documents forming a vocabulary of 4 570 938 tokens, routinely exceeded the 16 *gigabytes* of random access memory available on the author's computer.

Stars	Name	City	State	Average stars	Top category 1	Top category 2
4	Starbucks	Chandler	AZ	3.0	Food	Coffee & Tea
5	Culinary Dropout	Tempe	AZ	4.0	Restaurants	American (New)
1	Panda Express	Monroeville	PA	3.5	Restaurants	Fast Food
1	McDonald's	Madison	WI	1.0	Restaurants	Food

TABLE 10.4: *Selected structured attributes of the reviews from Table 10.3.*

10.3 The Twitter data set

The *Twitter* data set[4] was published along with the popular lexicon-based sentiment analysis model, *Vader* [131], and comprises 4 200 randomly selected microblogs (commonly referred to as "*Tweets*") from the social media site *Twitter*, along with a sentiment score assigned to each Tweet by a team of trained human annotators, as described in [131].

In order to transform the sentiment scores for each Tweet into distinct classes, the scores were first normalised to the interval $[-1, 1]$ and then classified into sentiment categories by applying threshold values of -0.05 and $+0.05$. More specifically, Tweets with normalised ratings below the negative threshold were classified as *negative*, those with normalised ratings above the positive threshold were classified as *positive*, and the remainder were classified as *neutral*. A similar process was followed by Hutto and Gilbert in the original paper employing the data [131]. An excerpt of the resulting data set, which was employed for the validation studies, is given in Table 10.5.

Tweet	Sentiment
Somehow I was blessed with some really amazing friends in my life who love me and send encouragement when I'm not feeling awesome. So lucky.	positive
LMAO, AMAZING!	positive
bad mood. :'(	negative
Yep, AT&T's customer service is terrible. Terrible, terrible, terrible. UGH.	negative

TABLE 10.5: *An excerpt of the Twitter data set (sic).*

10.4 On the versatility of the validation suite

A summary of the data sets employed to validate the ECCO framework, which include the data sets described in §10.1–10.3, as well as the data employed for the case study in Chapters 8 and 9 (henceforth referred to as the *SMS* data set), is given in Table 10.6. As shown in the table, each data set originates from a distinct domain. Two data sets represent binary sentiment classification problems (with classes *positive* and *negative*), whilst the remaining two represent ternary classification problems (with classes *positive*, *neutral* and *negative*). Furthermore, in two cases, supplementary structured data are also available.

[4] Available at https://github.com/cjhutto/vaderSentiment.

Data set	Domain	Problem type	Supplementary data
PL04	Movie Reviews	Binary	—
Yelp	Business Reviews	Binary	Business & user data
Twitter	Social Media	Ternary	—
SMS	Banking Survey	Ternary	Branch & customer data

TABLE 10.6: *A summary of the validation study data sets.*

Additional metadata on the validation study data sets are shown in Table 10.7. As is evident from this table, the data sets are not only diverse in domain, but also in class distribution, document length and vocabulary size $|V|$, defined as the number of types or unique tokens present in the original data set. Whilst the PL04 and Yelp data sets represent relatively balanced problems, sentiment is skewed towards the *positive* and *negative* classes for the Twitter and SMS data sets, respectively. The *neutral* class is strongly under-represented in the Twitter data set, whereas the minority classes are more equally distributed in the SMS data set. The PL04 data set exhibits the largest vocabulary size and mean document length, whilst the Twitter and SMS data sets exhibit the smallest vocabulary size and mean document length, respectively.

Data set	# Documents	Class distribution			Document length			$\|V\|$
		Positive	Negative	Neutral	Mean	Min	Max	
PL04	2 000	50%	50%	—	747.96	18	2 680	46 462
Yelp[5]	3 801	57%	43%	—	90.53	1	1008	15 879
Twitter	4 200	67%	27%	5%	14.07	1	78	11 468
SMS	2 500	11%	73%	16%	13.74	1	32	12 448

TABLE 10.7: *Metadata on the selected data sets.*

Although the case study examples all constitute relatively small data sets (of at most 4 200 documents), all of which were predominantly composed in English, there is considerable diversity in domain, medium (SMS, social media, review sites), document length and vocabulary size across the case study examples. Furthermore, the SMS data were gathered within an African context and therefore exhibit linguistic nuances distinct from the benchmark data sets, as well as a number of documents composed in African languages. These conditions were deemed sufficient to support the claim of generality of the framework applicability.

10.5 Chapter summary

In this chapter, three benchmark data sets from the literature, namely the PL04 data set, the Yelp data set and the Twitter data set, were described, along with the processes followed to prepare these data for use as input to the ECCO system. Along with the data set employed in Chapters 8 and 9 — the SMS data set — these data sets form the *validation suite* in respect of which the ECCO framework is validated in the next chapter. The versatility and suitability of this validation suite were discussed briefly at the end of this chapter.

[5]These data reflect properties of the reduced Yelp data set.

CHAPTER 11

Validation analysis

Contents

In this chapter, the ECCO framework is validated in respect of its general applicability and utility. The aim of this validation study is twofold. First, the ability of the framework to achieve its design objectives across various domains is evaluated. Secondly, the classification performance of the models developed by means of the sentiment modelling component of the framework are compared with the results of other researchers, where applicable. To this end, each of the three primary components of the framework are applied to the data sets in the validation suite introduced in the previous chapter by means of the ECCO system presented in Chapter 7. This is followed by a discussion of the results achieved *via* the framework, as well as selected comparisons with existing frameworks from the literature.

In contrast to the detailed demonstration of the framework's practical application in Chapter 9, the purpose of this chapter is to showcase the framework's ability to generalise across domains. The following case studies are therefore presented at a higher level with a greater focus on *the big picture*.

11.1 General application of the processing component

As described in §7.2.1, the *effect summary* returned to the user during the preprocessing stage, shown in Figure 6.4, was implemented in the form of a word cloud representation of the most frequent terms in the corpus, as well as a summary of the number of unique tokens present in the data before and after preprocessing in the ECCO system. In this manner, the user can effectively gauge the changes made to the data, both in the general and in the specific case.

Additionally, the classification accuracy of the smallest, computationally least expensive model considered in this study, namely the naïve Bayes model, was also evaluated for various preprocessing settings in order to gauge the relative effect of preprocessing on model performance. This was achieved by means of the *develop models* page of the ECCO system. More specifically,

a unigram presence bag-of-words model was selected as the input feature configuration and a three-fold cross validated grid search in respect of accuracy was performed to select the value for the hyperparameter α from the default range implemented in the ECCO system (see Table 9.2). The resulting mean test accuracy scores for two replications of this process are shown in Table 11.1 for all four data sets. For each replication, a different subset comprising 20% of the data was employed as the test set. Selected preprocessing configurations are highlighted in bold.

Preprocessing steps	PL04		Yelp		Twitter		SMS	
	$\|V\|$	CA	$\|V\|$	CA	$\|V\|$	CA	$\|V\|$	CA
None	**46 462**	**0.7915**	15 879	0.8755	11 468	0.8095	12 448	0.8200
2, 3, 4.1	45 860	0.7900	12 501	0.8930	9 173	0.8045	9 404	0.8240
2, 3, 4.1, 4.3	41 486	0.8010	11 349	0.8895	**8 462**	**0.8090**	8 948	0.8300
2–4	27 866	0.7565	**8 795**	**0.8915**	6 722	0.7875	**4 948**	**0.8460**

TABLE 11.1: *The effects of various preprocessing steps (numbering from Figure 6.9) on the vocabulary size $|V|$, as well as the classification accuracy (CA) of the naïve Bayes model trained in respect of unigram term presence features. Reported accuracy scores reflect the mean test accuracy of two models with randomly generated 80%-train 20%-test splits.*

At first, no preprocessing was applied. In the second experiment, stopword and punctuation removal, the grouping of tokens with primarily numerical characters into a single token *_num* and case normalisation were performed. Since the `NLTK` stop word list employed in the ECCO system is not tailored to a sentiment analysis problem, as mentioned in 7.2.1, certain words were excluded from this removal list. These include negation words, intensifiers such as *too* and *very*, and the exclamation mark. Furthermore, the numbers 1, 2 and 3 were preserved in the case of the SMS data set, as described in §9.1. In the third experiment, lemmatisation was applied. As before, stemming algorithms were not employed in order to avoid creating an unfair disadvantage for lexicon-based models in the subsequent modelling step, since these models function by comparing the words in a document to words in a pre-compiled lexicon (which do not typically contain word stems). Finally, in the last experiment, Algorithm 7.1 was applied in order to perform context-aware spelling correction.

In general, applying the various proposed preprocessing steps led to an increase in classification accuracy and a decrease in vocabulary size. In respect of the Yelp and SMS data sets, for example, superior performance was achieved by the model adopting the last preprocessing configuration, which resulted in a feature space 45% and 60% smaller than the original feature space, respectively. For the PL04 and Twitter data sets, however, the application of the spelling correction algorithm had a negative effect on model accuracy. This is likely because the spell checking library did not recognise several domain-specific expressions, such as *LOL* in the Twitter data set or *run-of-the-mill* in the PL04 data set. For the Twitter data set, the spelling algorithm was therefore not applied. For the PL04 data set, it was decided that no preprocessing would be applied, since the data set was already processed by the original authors. This, furthermore, allowed for a direct comparison of model results with benchmarks during the sentiment modelling stage.

The iterative application of preprocessing steps and utilisation of an effect summary, as per the ECCO framework, allowed for the selection of the best preprocessing configuration for each of the data sets, which was not necessarily the same in each case. Furthermore, this approach offered an initial insight into the data set, in terms of its vocabulary size, baseline performance

and word frequency distribution, as is illustrated by the word cloud representations in the case of the Yelp data set in Figure 11.1, as well as in the case of the SMS data set in Figures 9.3 and 9.4.

(a) *Unprocessed corpus*

(b) *Processed corpus*

FIGURE 11.1: *A word cloud illustrating the most frequent tokens in (a) the original, unprocessed corpus and (b) the final, preprocessed corpus of the Yelp data set.*

From Figure 11.1, it is evident that stop words such as *and*, *the* and *to* are the most frequent words in the unprocessed corpus of the Yelp data set. In the processed corpus, on the other hand, apart from retained stop words such as *but* and *not*, and the aggregated token *_num_*, words typically associated with business reviews constitute the most frequently observed words in the corpus. Interestingly, frequently observed words, which include *food*, *order*, and *service*, appear to be associated with reviews of eating establishments, in particular.

In summary, the preprocessing component of the ECCO framework was applied to reduce the vocabulary of each corpus *via* filtering and normalisation without compromising model accuracy, and to bring information-bearing words to the forefront. Furthermore, the user was able to customise and selectively apply each of these processes in order to tailor the preprocessing approach to the data set in question.

11.2 General application of the modelling component

As previously mentioned, the modelling component of the ECCO framework is primarily concerned with enhancing and improving the MST selection process. In the validation case studies decribed in this section, a similar approach was followed as that adopted in §9.2. More specifically, the naïve Bayes, SVM, logistic regression and ANN algorithms were implemented, taking nine variants of the bag-of-words representation as input (all possible combinations of the term presence, term frequency and TF-IDF document models with the n-gram ranges $(1, 1)$, $(1, 2)$ and $(2, 2)$). A manual search was conducted by means of the *Tensorboard* interface in order to

narrow the search space for the ANN, CNN and LSTM models. Subsequently, the hyperparameters of all machine learning models were tuned employing the resulting hyperarameter grids given in Table 11.2.

Algorithm	Hyperparameter	Values			
		PL04	Yelp	Twitter	SMS
Naïve Bayes	α	0.0001, 0.2, 0.4, 0.6, 0.8, 1			
SVM	C	0.01, 0.1, 1, 10			
	γ	0.001, 0.01, 0.1, 1			
	Kernel	linear, radial, sigmoid, P1, P2, P3			
Logistic regression	C	0.01, 0.1, 1, 10			
	Solver	SAG, Newton-CG, LBFGS			
	Max iterations	100			
ANN	Neurons per hidden layer	100; 100, 100; 30	10,10	50, 50	10, 10
	Regularisation	ℓ_2, None	ℓ_2	ℓ_2	ℓ_2
	λ	0.2, 0.3	0.01	0.002	0.001
	Dropout probability	0, 0.5	0	0	0
	Number of epochs	10, 30	10	10	10, 12
	Initial learning rate	0.001	0.001	0.001	0.02
	Learning rate decay	0			
	Batch normalisation	No			
	Activation function	ReLU			
	Loss Function	Cross-entropy			
	Solver	ADAM			
CNN	Embedding size	10			
	Convolution type	Valid			
	(Kernel size, stride, #filters)	(3, 1, 1); (1, 1, 10)	(4, 1, 12)	(3, 1, 12)	(1, 1, 20)
	Pooling kernel size	(2, 2)	—	(2, 2)	—
	Regularisation	ℓ_2, None	ℓ_2	ℓ_2	ℓ_2
	λ	0.001, 0.0001	0.001	0.02	0.01
	Number of epochs	25	10	10	10, 12
	Initial learning rate	0.001, 0.0001	0.01	0.01	0.01
	Learning rate decay	0			
	Batch normalisation	No			
	Loss Function	Cross-entropy			
	Solver	ADAM			
LSTM	Embedding size	10			
	LSTM output size	1	10	3	10
	Regularisation	ℓ_2	None	None	None
	λ	0.01, 0.02	—	—	—
	Number of epochs	10	10	5	4, 8
	Initial learning rate	0.001	0.01	0.03	0.01
	Dropout probability	0			
	Learning rate decay	0			
	Loss Function	Cross-entropy			
	Solver	ADAM			

TABLE 11.2: *The hyperparameter grids employed in the validation study.*

Due to the increased number of experiments to be executed in total, as well as the larger vocabulary sizes and resulting term-document matrices for some of the data sets in the validation suite, a variation of a random search (see §3.2.1) comprising ten iterations was implemented instead of the exhaustive grid search, where combinations of parameter values were selected randomly

from the grid during each iteration[1]. Since the number of points on the hyperparameter grid exceeded ten only in the case of the SVM and logistic regression models, as well as in selected cases for the deep learning models, however, a grid search was effectively performed in all other cases.

The procedure applied in §9.2 to find a vocabulary size at which the performance begins to reach a plateau was retained for the SMS data set. In order to allow for a fair comparison between the performance achieved by the models trained according to the ECCO framework and benchmark results, however, the full vocabulary was employed for the PL04, Yelp and Twitter data sets.

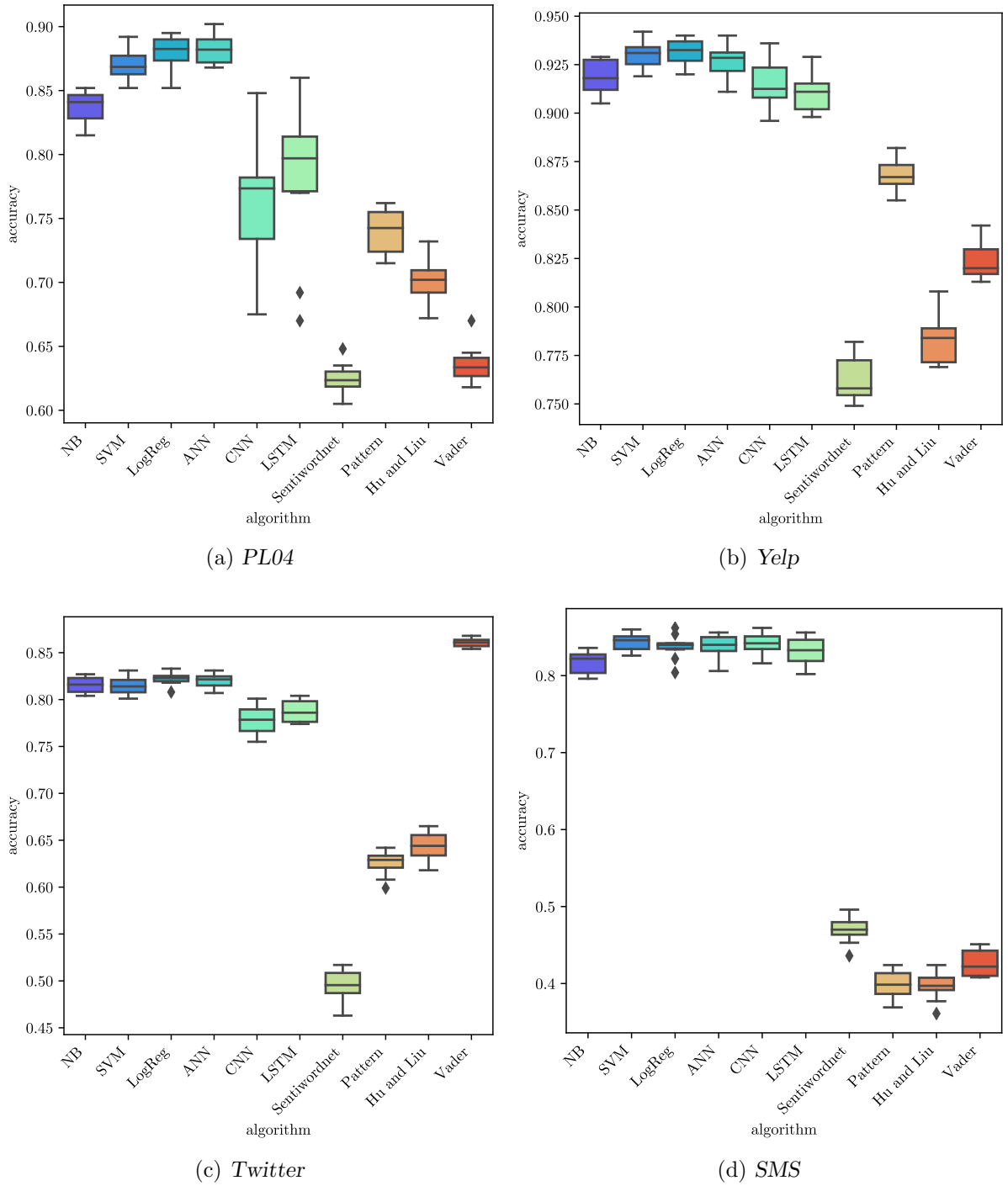
Finally, ten replications were performed to account for variability in training and test data, as well as network initialisation, as before. The training data comprised 70% of the available data, whilst the validation and test data comprised 10% and 20%, respectively. In addition to the machine learning algorithms of Table 11.2, the same pre-trained lexicon-based models employed in Chapter 9 were evaluated in respect of each data set in the validation suite. The accuracy scores achieved by each algorithm during the ten experimental runs are shown in Figure 11.2 for each case study data set. Where multiple models were trained by means of the same algorithm (*i.e.* where several different input data representations were employed as input), the best performing model was selected. Detailed results in respect of the PL04, Yelp and Twitter data sets may be found in Appendix C.

It is evident from the figure that most, if not all, of the machine learning models trained according to the process outlined in the ECCO framework outperformed the off-the-shelf lexicon-based models by a large margin across all four domains. The CNN and LSTM algorithms fared slightly poorer than the remaining models for the PL04, Yelp and Twitter data sets. This is likely due to the specific model architecture chosen for the case study experiments, which includes a word embedding matrix trained end-to-end with the remainder of the network. With reference to Table 10.7, however, all of the data sets are relatively small, containing only 2 000–4 200 documents, compared to the large embedding matrices resulting from the vocabulary sizes of between 11 468 and 46 462 tokens. The machine learning models therefore likely did not have sufficient data to learn adequate embedding parameters. This effect is especially pronounced for the PL04 data set, which has both the largest vocabulary size and longest mean document length.

The best-performing models generated by means of the framework consistently achieved competitive results. In the original paper on the PL04 data set [230], Pang and Lee achieved an accuracy score of 87.15% using an SVM classifier. Using the MST selection process outlined by the ECCO framework, the same model achieved scores of up to 89.2% during the ten experiments, with a mean accuracy score of 86.96%. The ANN model, on the other hand, achieved a mean accuracy of 88.27%. State-of-the-art results on the data set, as published by Nguyen *et al.* [215] and Maas *et al.* [183], range from 86.2%–91.6%.

Whilst most researchers employing the Yelp data set predicted star ratings directly, the data set was transformed into a binary sentiment classification problem in this study, as was done by Salinca [267], who achieved a maximum accuracy score of 92.6%, using a subset of 10 000 of the data points and evaluating the model in respect of 20% of this subset. The best model (logistic regression) trained according to the ECCO framework in respect of the extracted subset of the data achieved an accuracy score of up to 94% during the ten experiments, with a mean accuracy score of 93.15%.

[1]The notion of a parameter grid was retained for the sake of simplicity and uniformity of data input for the user. The use of a range input with a random search is discussed as future work later in this dissertation.



(a) *PL04* (b) *Yelp*

(c) *Twitter* (d) *SMS*

FIGURE 11.2: *Model performance in terms of accuracy. The accuracy values achieved by the best model-feature pairs during ten experimental runs are shown in the form of a box plot for each of the tested models and each data set in the validation suite.*

Finally, in respect of the Twitter data set, the best-performing ECCO-trained model was logistic regression with a mean accuracy score of 82.2%. This was significantly lower than the mean score of 86.06% achieved by Vader[2], which was tailored specifically to this domain and data set. Given this fact, however, the results were still deemed competitive.

The lexicon-based models, on the other hand, achieved highly variable results across domains. Whilst the Vader model dominated in respect of social media data, it achieved the second-poorest results in the movie review domain. In fact, in each of the four domains, one of the four lexicon-based models performed better than the remaining three in turn. The difference in performance between machine learning and lexicon-based models is particularly pronounced for the SMS data set, as was already illustrated in §9.2, where language is influenced by a non-US context.

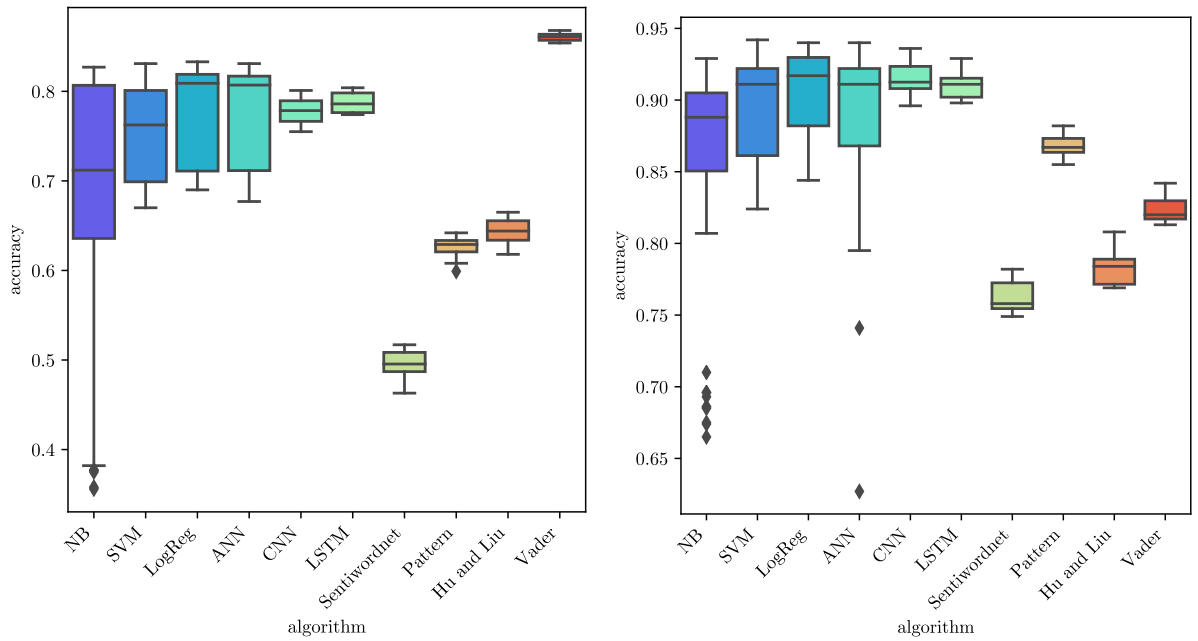
The findings of §9.2 are therefore not limited to the banking or SMS domain, or a South African context, but are presented across domains for text data conveyed *via* various media and with varying linguistic styles. This further corroborates the approach adopted in the ECCO framework, whereby several models are developed and compared for each new data set, as opposed to the application of a specific model, which may achieve state-of-the-art results in one domain, but is unlikely to match this performance in other domains. In the ECCO framework, the emphasis is on *consistently good* rather than *conditionally excellent* performance.

Simply training a new model in respect of the available data, however, is not sufficient. Although other studies have shown machine learning models to outperform lexicon-based models in respect of certain data sets [111], such results are only achievable with the correct combination of features, model parameters and hyperparameters (the MST). As mentioned in Section 6.1, however, other existing frameworks do not account for this process and are therefore of little practical use for users who are not well versed in the MST selection processes. To emphasise this point, consider Figure 11.3, in which the performance of the machine learning algorithms in respect of two of the data sets is shown for *all* model-feature combinations as opposed to the selected MST in each experimental run, as shown in Figure 11.2. Comparing this figure with the previous one highlights the importance of good feature engineering. Whilst the performance of the naïve Bayes algorithm, for example, ranged between 80.4% and 82.7% for the selected feature sets in Figure 11.2(c), the same model's performance varied between 35.6% and 82.7% considering all feature sets in Figure 11.3(a). More specifically, the model achieved a mean accuracy of 81.31% using the unigram presence feature representation, and a mean accuracy of 38.06% using the bigram TF-IDF feature representation. Selecting the correct features is therefore of critical importance to the performance of a machine learning algorithm.

As with the performance of a particular model, however, the relative performance of a particular set of features is not necessarily consistent across all data sets. Consider, for example, the variation in relative performance between the TF-IDF document model and the n-gram range $(2, 2)$ (bigrams only) across the PL04 and SMS data sets illustrated in Figure 11.4. Whereas the bigrams representation fares particularly poorly in respect of the SMS data set (as was already discovered in §9.2), this difference is not as pronounced for the PL04 data set (and may even be reversed for a different data set). Similarly, whilst the performance of the TF-IDF representation is relatively inconsistent in respect of the PL04 data set, this is not the case for the SMS data set where this representation, in fact, achieves marginally superior performance.

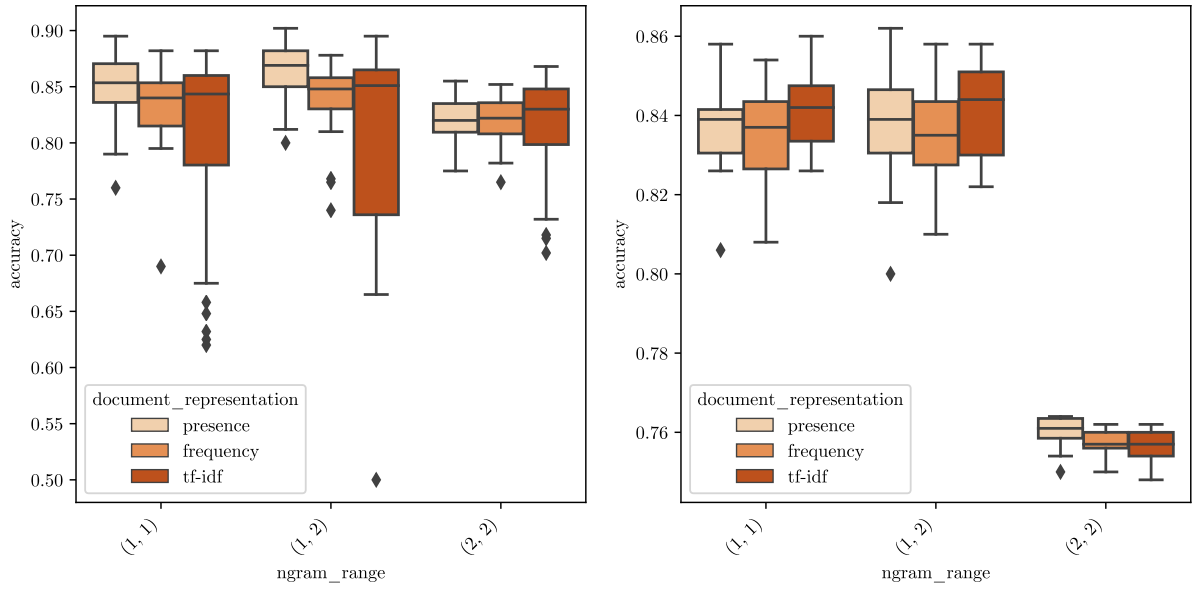
The examples above relate to the variation in performance caused by feature engineering efforts. The effect of hyperparameter selection is, however, not included in these comparisons, since

[2]In spite of best efforts to imitate the method of implementation, testing conditions and metrics used in the original paper [131], the reported F1 score of 96% could not be replicated. Relevant implementation details adopted in [131] may be missing to this end.



(a) *Twitter* (b) *Yelp*

FIGURE 11.3: *Performance variation induced by feature engineering. The accuracy values achieved by all model-feature pairs during ten experimental runs are shown in the form of a box plot for each of the tested models in respect of (a) the Twitter data set and (b) the Yelp data set.*



(a) *PL04* (b) *SMS*

FIGURE 11.4: *Performance for various feature sets. The accuracy values achieved during ten experimental runs are shown in the form of a box plot for all models using a particular feature set in respect of (a) the PL04 data set and (b) the SMS data set.*

a grid or random search was performed for each experiment in the ECCO system. From the outputs of the search algorithm, however, observations were made as to the significant effect of this process on model performance. Within the single model-feature combination of the logistic regression model with a unigram presence representation, for example, the validation performance[3] achieved by models employing different hyperparameter combinations varied between 89.24% and 96.91% for the Yelp data set. The final hyperparameter configuration included a value of 1 for the tuning parameter C and the use of the *stochastic average gradient* solver. For the unigram TF-IDF representation, on the other hand, the *Newton-CG* solver with $C = 10$ led to the best performance.

It is therefore imperative to employ a structured approach towards finding suitable sentiment models for each new data set in order to achieve consistent performance across various domains. To the best of the author's knowledge, the ECCO framework is the only framework which integrates the process of selecting a suitable MST into a sentiment analysis framework, thereby offering a practical guide to researchers and analysts on how to robustly develop competitive models for sentiment analysis across various domains and problem types.

Finally, with respect to Process 12.0 of the ECCO framework, the combination of models in an ensemble, the same three ensemble pruning approaches (greedy, model and model ML) were applied in conjunction with the same six ensemble configurations (DS, DW, DM, SS, SW, SM) employed during the case study in §9.2.3.

For the PL04, Yelp and Twitter data sets, the number of selected base learners obtained by maximising the optimisation problem in (9.2)–(9.3) by means of a genetic algorithm was two, both when considering only machine learning models and when considering the entire pool of candidate base learners. As mentioned in §9.2.3, three base learners were selected for the SMS data set, by means of both the model and model ML selection approaches. A summary of the selected ensemble configurations for all four data sets in the validation suite is shown in Table 11.3 along with their associated diversity, accuracy and favourability metrics. Details about individual base learners can be found along with the results of each ensemble configuration in Appendix D.

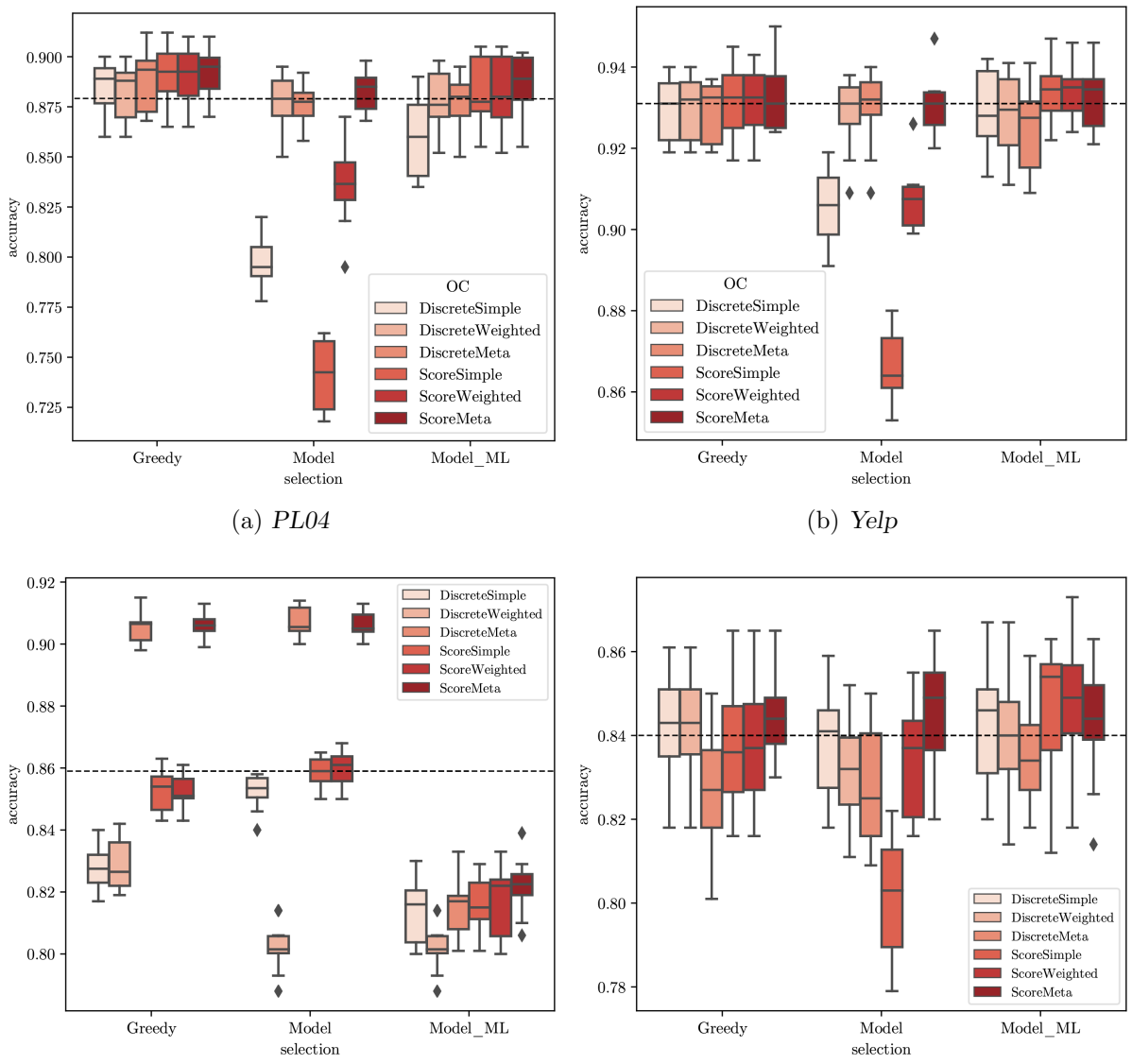
Data set	Selection approach	K	Diversity	Accuracy	Favourability
PL04	Greedy	5	0.3500	0.8721	0.5683
	Model	2	1.0000	0.8100	0.8192
	Model ML	2	0.7500	0.8677	0.7332
Yelp	Greedy	5	0.4000	0.9265	0.6088
	Model	2	1.0000	0.8985	0.8591
	Model ML	2	0.7500	0.9257	0.7593
Twitter	Greedy	5	0.6000	0.8264	0.6538
	Model	2	1.0000	0.8401	0.8328
	Model ML	2	0.7500	0.8157	0.7098
SMS	Greedy	5	0.3500	0.8529	0.5532
	Model	3	0.9167	0.7199	0.7436
	Model ML	3	0.7500	0.8409	0.7238

TABLE 11.3: *The results of the ensemble selection approaches applied to each of the validation data sets.*

The accuracies achieved by each of the examined ensemble configurations in respect of each of the validation data sets during the ten experimental runs are illustrated in the form of box plots

[3]Here, the three-fold cross validated accuracy in respect of the training data.

in Figure 11.5. As before, the median performance achieved by the best base learner is denoted by the black dotted line. In general, median performance improvements over the best individual model were observed for several ensemble configurations in respect of all four validation data sets. The most significant improvement of 5.53% was achieved in respect of the Twitter data set, whilst only a marginal improvement of 0.43% was achieved in respect of the Yelp data set, as is summarised in Table 11.4.



(a) *PL04* (b) *Yelp*

(c) *Twitter* (d) *SMS*

FIGURE 11.5: *Box plots of the accuracy achieved by ensembles in respect of the adopted ensemble pruning approach, output type and combination method employed during ten experimental runs. The median accuracy achieved over ten experimental runs of the best base learner is denoted by the dashed black line.*

The model selection approach, which is associated with the largest ensemble diversity for all four data sets, resulted in the widest range of performance results across the six ensemble

[4]In this case, the best base learner refers to the best individual model across all three sets of base learners, irrespective of whether the model appeared in all three sets. The indicated improvement was measured relative to this value.

		Best ensemble			Best relative
Data set	BBL	Greedy	Model	Model ML	improvement
PL04	87.90	**89.50**	88.50	88.90	1.82%
Yelp	93.10	93.25	93.20	**93.50**	0.43%
Twitter	85.90	**90.65**	90.55	82.25	5.53%
SMS	84.00	84.39	84.90	**85.40**	1.68%

TABLE 11.4: *The median test accuracies (in percentages) achieved by the best base learners*[4] *and best ensemble configurations for each ensemble selection approach in respect of each of the validation data sets. The performance of the best ensemble is highlighted in bold.*

configurations — some achieving significantly lower accuracy scores than the best base learner, and others significantly outperforming this benchmark. The combination of scoring outputs by means of meta-learning, however, consistently resulted in superior results for this selection approach compared with the performance of the best individual model. Even with an average base learner accuracy 14% lower than that of the best individual model, as was the case for the SMS data set, this ensemble configuration resulted in a 1% increase in median accuracy[5]. These results suggest that there is potential for selecting base learners according to the proposed metric of ensemble diversity in combination with a meta-learning approach.

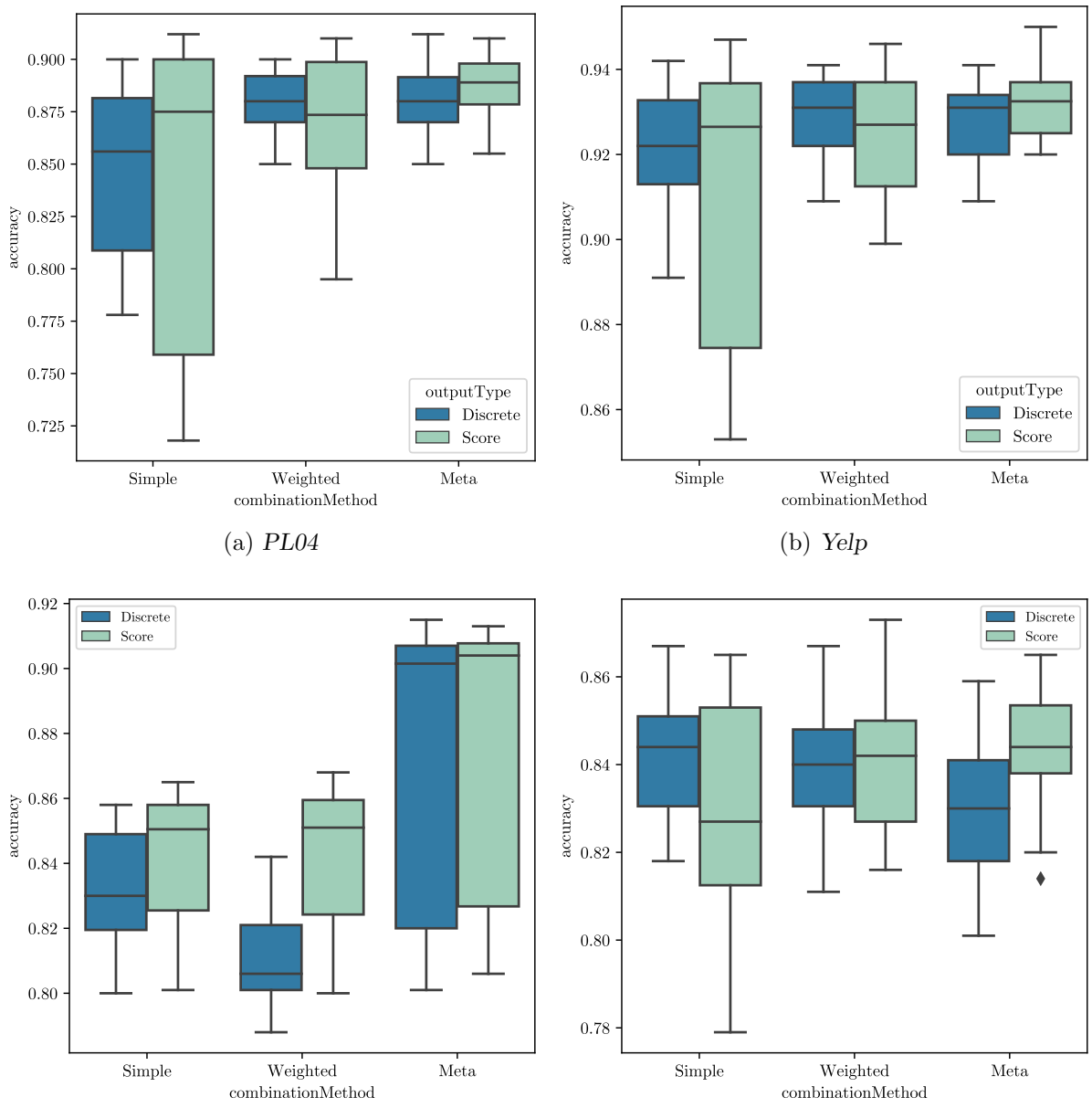
There does not, however, appear to be a direct correlation between the ensemble favourability scores in Table 11.3 and the scores achieved by the associated ensemble models summarised in Table 11.4. Whilst the most *favourable* set of base learners was always generated according to the model selection approach, this set did not result in the best ensemble performance in any of the four cases (although the results were typically very close to those of the best ensemble). The individual metrics of ensemble size, diversity and accuracy, when considered separately, were equally inadequate predictors of ensemble performance. Notwithstanding this, the largest improvement over the best base learner (observed in respect of the Twitter data set) was achieved by the most *diverse* ensemble of those generated according to the greedy approach, which generally produced the largest ensembles with high accuracy scores. A different weighting of ensemble size, diversity and accuracy in the proposed favourability metric may therefore be more effective in predicting ensemble performance.

It is also worth noting that the Twitter data set presented the only case in which a lexicon-based model was not significantly outperformed by all of the machine learning models. It was therefore also the only case where a high ensemble diversity could be achieved without a significant trade-off in ensemble accuracy. On the other hand, whilst the model ML selection approach effectively applied a *minimum* threshold on base learner accuracy for the other data sets in the validation suite, excluding the Vader model from the ensemble corresponded to setting a *maximum* threshold on base learner accuracy in respect of the Twitter data set. It is, therefore, unsurprising that the ensemble selected by the model approach achieved better results than those of the ensemble selected *via* the model ML approach only in respect of this data set. The inclusion of a constraint on the minimum base learner accuracy in the optimisation problem of (9.2)–(9.3) may present an additional avenue for improvement of the proposed ensemble selection approach.

With respect to the effectiveness of the output types and combination methods employed in the ensemble models, a clear trend was observed across all four data sets, as shown in Figure 11.6. More specifically, the combination of scoring or probabilistic outputs by means of meta-learning

[5]In both cases, scores are reported relative to the median accuracy of the best base learner.

matched or outperformed the remaining five ensemble configurations in respect of all the data sets in the validation suite. The relative performance among the other ensemble configurations varied across data sets. In nine out of twelve cases, however, ensembles employing scoring outputs outperformed those utilising discrete outputs.



(a) *PL04* (b) *Yelp*

(c) *Twitter* (d) *SMS*

FIGURE 11.6: *Box plots of the accuracy achieved by ensembles in respect of the adopted ensemble pruning approach, output type and combination method employed during ten experimental runs. The median accuracy achieved over ten experimental runs of the best base learner is denoted by the dashed black line.*

In summary, the *complementary* selection of MSTs proposed in the ECCO framework resulted in a median performance improvement over *competitive* selection across all four data sets. Furthermore, it was shown that ensembles formed by combining pre-trained models with those trained specifically in respect of the available data can perform better than their best compo-

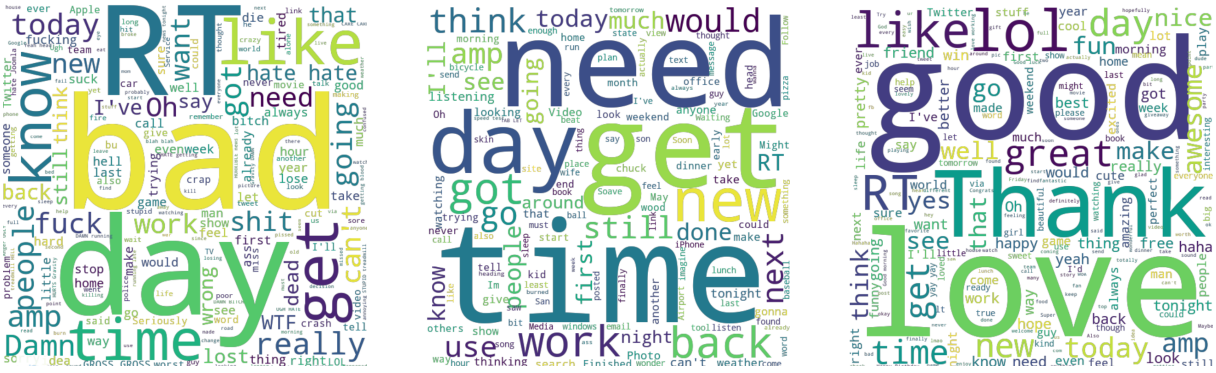
nent model, and significantly better than the average performance of their constituent models when a meta-learning approach is adopted.

11.3 General application of the analysis component

For each of the case study data sets, the best-performing individual model was selected in order to proceed to the analysis phase of the ECCO framework[6]. More specifically, the ANN, logistic regression and Vader models were selected to classify the PL04, Yelp and Twitter data, respectively, whereas the CNN model was selected for the SMS data set, as described in §9.3. The ECCO system was then employed to analyse the results of these models in order to extract valuable insights from the data.

In §9.3, the application of each of the modules of the analysis component of the framework was described in detail. In this section, on the other hand, illustrative examples are provided in order to showcase the ability of these modules to generalise to across all four of the domains investigated.

As previously mentioned, Processes 16.0 and 15.0 of the analysis component of the framework (shown in Figure 6.8) were applied to the case study data by means of summaries, including a textual summary (by *template instantiation*), general statistics, and word cloud representations of documents in each sentiment class, as well as by an LDA topic analysis. The word cloud representations of the documents classified into each of the three sentiment classes for the Twitter data set, for example, are shown in Figure 11.7. From this representation, it is clear that the terms *bad*, *day* and *RT* are the most common terms mentioned in negative Tweets, whilst the terms *good*, *thank* and *love* are most commonly mentioned in positive Tweets. Interestingly, the expression *RT* or *retweet*, referring to the re-posting or sharing of another person's Tweet, is more prominent in negative rather than positive Tweets in this data set. This may allude to the notion that *'bad news travels fast.'*



(a) *Negative* (b) *Neutral* (c) *Positive*

FIGURE 11.7: *Word cloud representations of the most frequently used words in each sentiment class for the Twitter data set.*

Similarly, the word cloud representations of the PL04 and Yelp data sets shown in Figure 11.8 also give some initial insight into the data in both cases. The terms *order* and *location*, for instance, appear to be more prominent in negative reviews than in positive reviews for the

[6]As in Chapter 9, the relative increase in accuracy of the ensemble models was not deemed large enough to significantly alter the classification results for the analysis phase.

Yelp data set. Interestingly, the term *place* is, on the other hand, frequently encountered in positive reviews. Reviewers therefore likely speak about a *great place* in the positive sense, but tend to complain of an establishment's unfavourable *location*. In the case of the PL04 data set, similar word usage is exhibited across sentiment classes, with *movie*, *character* and *one* being the most prominent terms in both cases. Even the polarised term *good* is prevalent on both sides, which may suggest that reviewers discuss similar aspects of the film in both positive and negative reviews. The term *plot*, however, appears prominently only in the word cloud of negative reviews.

(a) *Yelp; negative* (b) *Yelp; positive*

(c) *PL04; negative* (d) *PL04; positive*

FIGURE 11.8: *Word cloud representations of the most frequently used words in the positive and negative sentiment class for the Yelp data set in (a) and (b), and for the PL04 data set in (c) and (d), respectively.*

LDA topic models were visualised by invoking the *LDAvis* package. Based on both the frequency with which each of the most salient terms in the corpus were observed in a given topic and the relevance of words to a given topic, both of which are visualised in the *LDAvis* interface, several important keywords could be associated with each topic. The generated topic model of the SMS data set shown in Figure 9.34 revealed five clusters of frequently co-occurring words in the corpus, identified as topics pertaining to, for instance, loan applications or ATMs. The same approach could be applied to gain further insight into the remaining data sets.

The visualisation of the topic model for the Yelp data set, for example, is shown in Figure 11.9. This model was generated based on five topics and five passes through the available data. For Topic 1, for instance, which is selected in the figure, numerical expressions are most relevant and frequent, along with the terms *order*, *time*, *food*, *get*, *minute* and *wait*. These word co-occurrence patterns suggest that Topic 1 is primarily concerned with reviewers having had to wait long periods of time for their (food) orders.

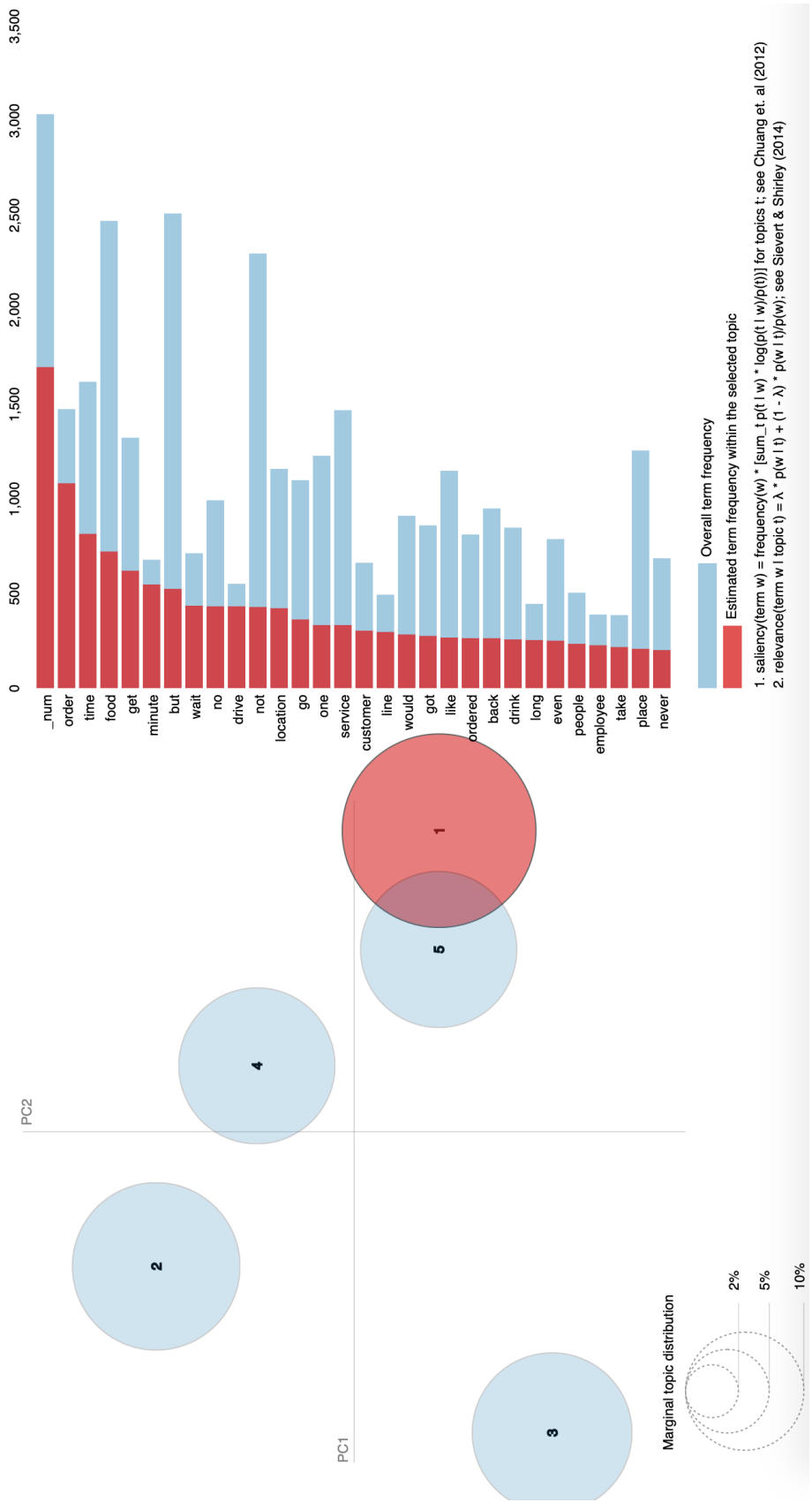


FIGURE 11.9: *LDAvis visualisation of the LDA topic model for the Yelp data set based on five topics and five passes through the data. Topic 1 is currently selected and its most relevant terms are shown on the right-hand side of the figure.*

Similarly, a five-topic model was generated for the PL04 data set, as shown in Figure 11.10. The most salient token in this topic is the quotation mark, followed by the semi-colon and the term *characters*. Given the significantly larger frequency of these tokens in the topic compared to the other salient tokens, the discussion of the film and its characters, in particular, appears to be shaped by quotations from the film, quoted titles of films or, perhaps, the ironic or euphemistic use of quotation marks (*e.g. "You've Got Mail is the definition of a 'cute' movie"*) in this topic. By viewing samples of reviews filtered according to the quotation mark by means of the ECCO system, the former two cases appear to be most common in the data set.

After having identified prominent topical keywords, their importance and relevance to each sentiment class were estimated by means of further visualisation. Several frequent keywords mentioned in the PL04 data set, for example, are illustrated in Figure 11.11. Since the class distribution of the data set was perfectly balanced, keywords which occur significantly more frequently in one sentiment class than in another appear to be related to this particular sentiment. From the figure, it is evident, for example, that the *plot* and *director* of a reviewed movie are significantly more likely to be mentioned in the criticism of a film rather than its acclamation.

Similarly, the analysis of the most frequent keywords automatically extracted by means of the noun phrase extraction process described in §7.2.3 for the Yelp data set is shown in Figure 11.12. Since this data set was also relatively balanced (with 43.2% of observations classified as *negative* and 56.8% classified as *positive* by the selected algorithm), it is immediately evident from the figure that the terms *order* and *manager* are strongly associated with negative reviews. Issues escalated to the manager, and problems with orders therefore frequently appeared to be the subject of negative reviews. Where strong class imbalances are present, a different interpretation of the resulting graphs is required. This was illustrated in §7.2.3, where the keywords *loan* and *ATM* were identified as some of the most pressing concerns for bank customers, who generally submitted primarily negative reviews.

Where supplementary data are available (as for the Yelp and SMS data sets), the relationship between these structured variables and sentiment can be explored using the ECCO framework. Such an analysis can provide valuable further insight into the data and expose hidden patterns useful for decision making. As mentioned in §6.1, the utility of most sentiment analysis frameworks is limited to the sentiment modelling phase or finding an accurate representation of the sentiment distribution (*e.g.* 70% of customers are dissatisfied). The analyses of the review text described above already offer helpful further insights (*e.g.* many dissatisfied customers complained about insufficient ATMs). This information is, however, often not yet specific enough to drive decision making. The additional analysis of structured variables can provide a third layer of insight (*e.g.* customers complaining about a lack of ATMs tend to be those in rural areas, and pay monthly fees that are 15% higher than the customer average), rendering the analysis more actionable (*e.g.* address the ATM issue by installing additional machines in rural areas or entering into partnerships with local supermarkets for inexpensive cash withdrawals). As with any such analysis, however, care must be taken when interpreting the results. The process outlined in the ECCO framework can only uncover hidden correlations — the assessment of possible causalities lies with decision makers.

Process 18.0 of the ECCO framework refers to type-specific visualisations of model-feature relationships. Box plots may, for example, be employed to visualise the relationship between quantitative variables and sentiment. In Chapter 9, for instance, box plot representations of customer age revealed insignificant differences in this variable across sentiment classes. In the case of the Yelp data set, significant differences were exhibited by some variables. From Figure 11.13, for example, it may be deduced that sentiment expressed in reviews correlated almost as strongly with the average star rating given by a user across establishments on the platform

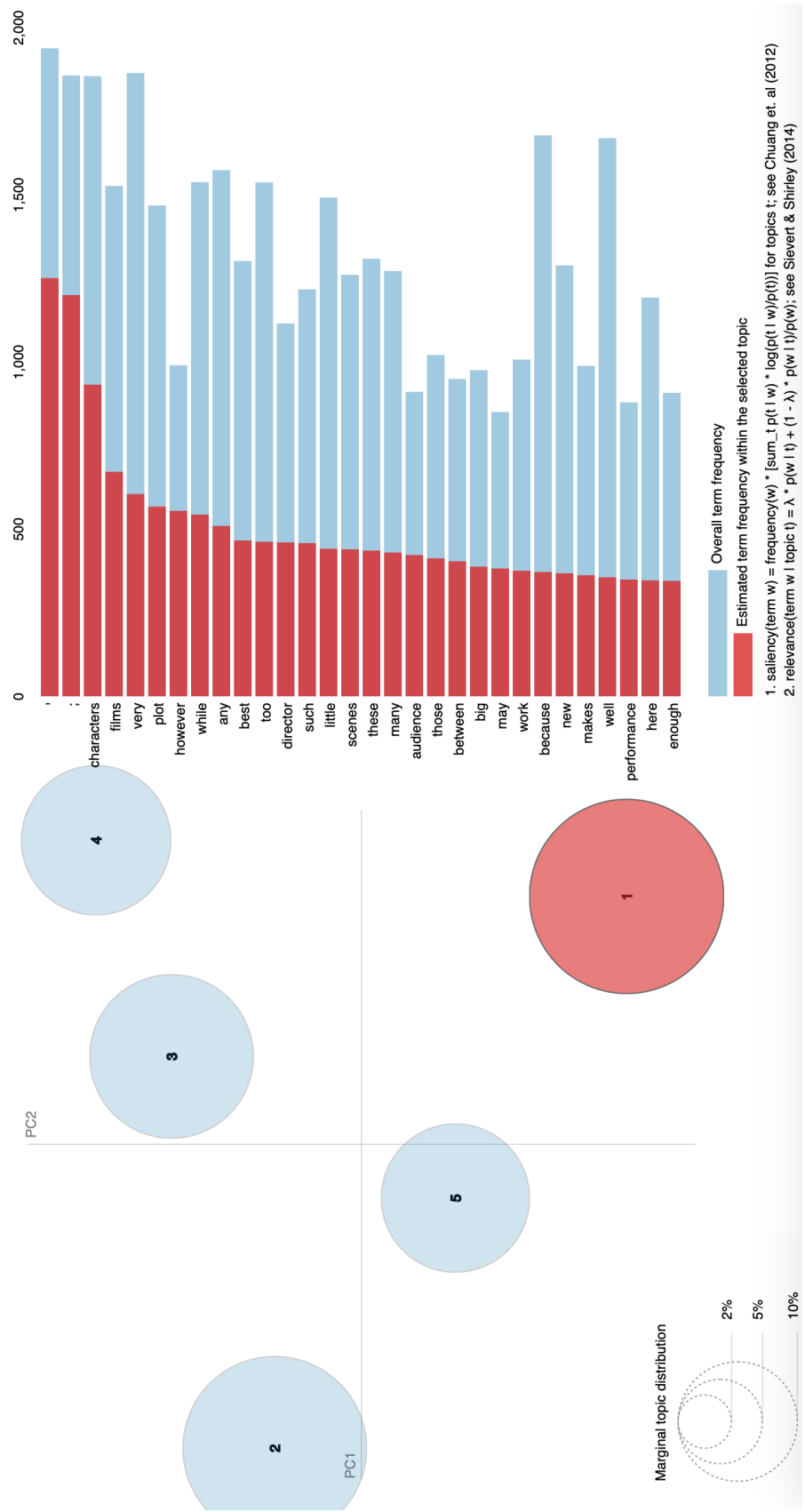
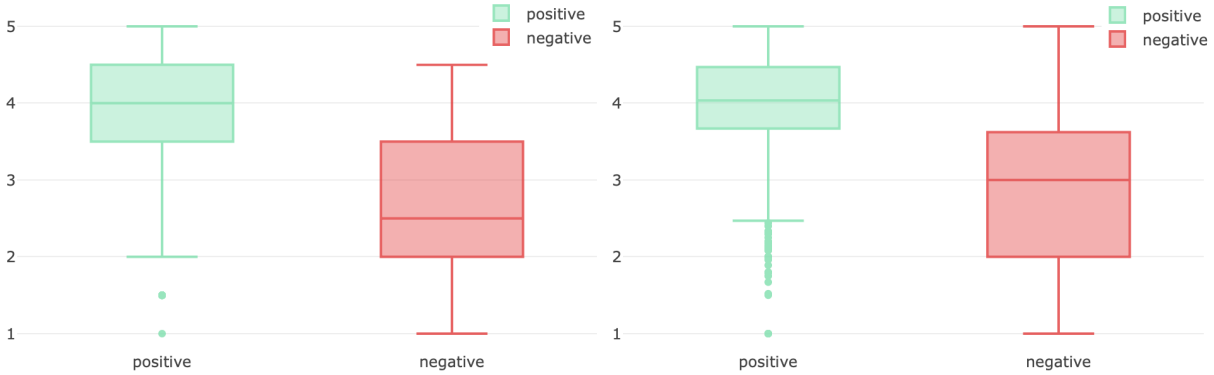


FIGURE 11.10: *LDAvis visualisation of the LDA topic model for the PL04 data set based on five topics and two passes through the data. Topic 1 is currently selected and its most relevant terms are shown on the right-hand side of the figure.*

FIGURE 11.11: *The frequency of various keywords in documents of each sentiment class for the PL04 data set.*

FIGURE 11.12: *The frequency of various keywords in documents of each sentiment class for the Yelp data set.*

as with the average star rating of an establishment. Both the experience and the attitude of a visitor may, therefore, influence the nature of a review.

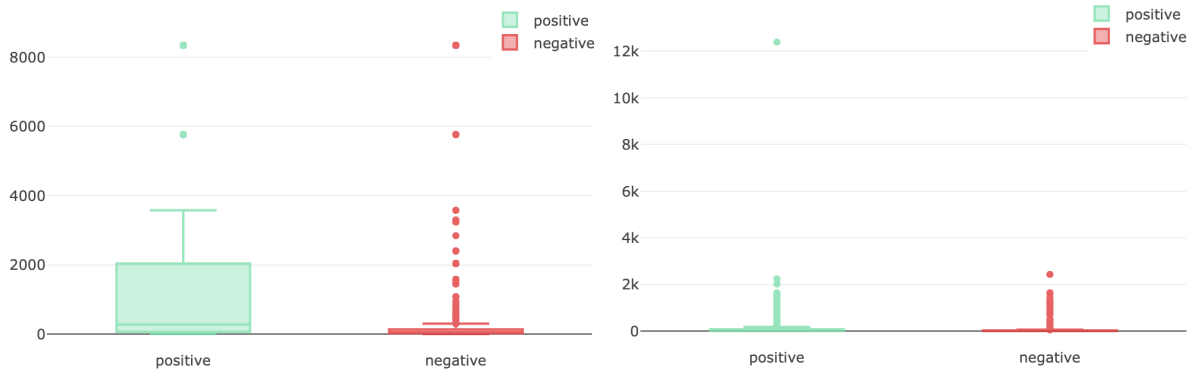


(a) *Average stars establishment* (b) *Average stars user*

FIGURE 11.13: *Box plots illustrating the relationship between sentiment and (a) the average star rating received by an establishment or (b) the average star rating given by a user.*

Conversely, the typical review count of an establishment subject to positive reviews in the data set is significantly larger than that of establishments subject to negative reviews, with an interquartile range of $[67, 2\,041]$ for positive reviews and $[24, 137]$ for negative reviews, as shown in Figure 11.14(a). This is consistent with the findings of several studies that online reviews present a persistent positivity bias [37] and may suggest that disappointed customers are less likely to leave a review than satisfied customers. Users submitting positive and negative

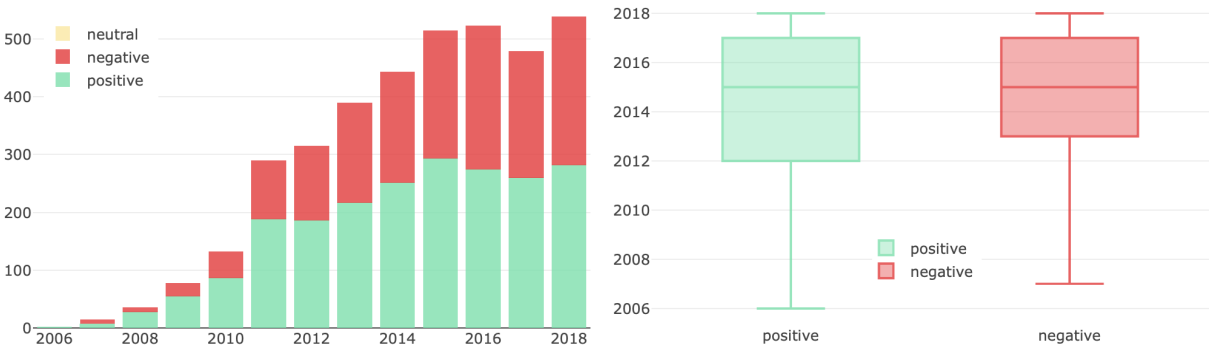
reviews, however, do not exhibit as pronounced disparities in individual review counts, with the interquartile range of review counts for users submitting positive and negative reviews equal to $[7, 75]$ and $[3, 26]$, respectively. Whilst it was shown in Figure 11.13(b) that users tend towards submitting either positive or negative reviews, both groups appear to do this close to equally often.



(a) *Review count establishment* (b) *Review count user*

FIGURE 11.14: *Box plots illustrating the relationship between sentiment and the review count of (a) the associated establishment and (b) the associated user.*

Similarly, histograms may be used to illustrate the relationship between sentiment and qualitative variables. The year in which a reviewer joined the Yelp review site could be viewed as both a quantitative and as a qualitative variable. As shown in the histogram in Figure 11.15(a), users who submitted reviews in the selected time frame (the year 2018) were more likely to have joined the review site in recent years. Interestingly, the proportion of negative reviews appears to increase among reviewers who joined the site more recently. The box plots in Figure 11.15(b), on the other hand, indicate that the median joining year is 2015, both for users who submitted positive reviews and users who submitted negative reviews.

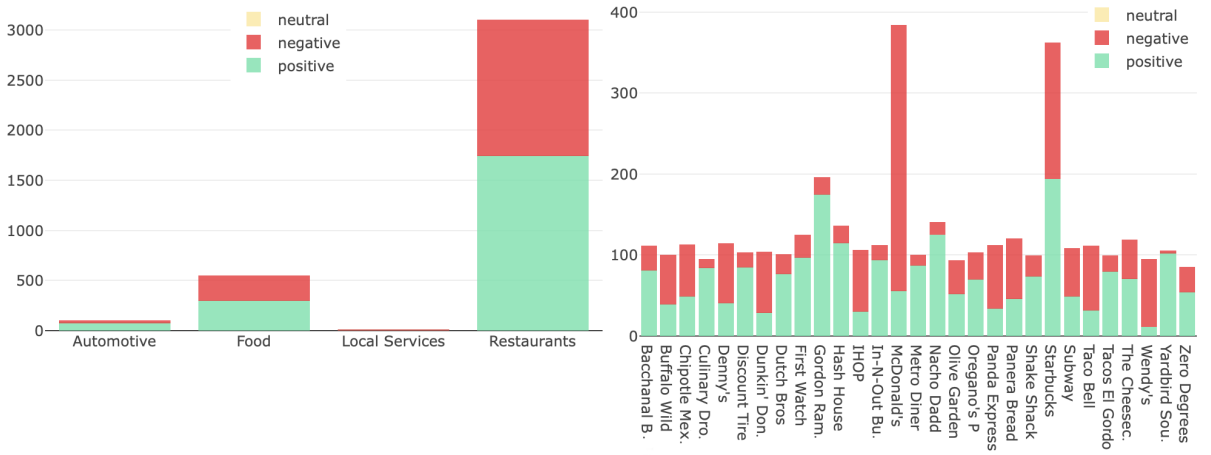


(a) *Qualitative view* (b) *Quantitative view*

FIGURE 11.15: *Sentiment by tenure. The year in which users who submitted positive and negative reviews is shown (a) in the form of a histogram and (b) in the form of a box plot.*

From Figure 11.16(a), it is clear that most of the reviews in the Yelp data set refer to eating establishments, as was already suspected based on the word usage illustrated in the word cloud in Figure 11.1. A closer look into the *names* attribute, shown in Figure 11.16(b), reveals that a

subset of 29 establishments was selected during the procedure applied to reduce the size of the data set. Most of these establishments are mentioned approximately 100 times in this reduced data set. This is consistent with the random selection of 10% of the reviews on each establishment that was mentioned more than 1 000 times in the original 2018 data set, as described in §10.2. The three most popular establishments are *McDonald's*, *Starbucks* and *Gordon Ramsay Hell's Kitchen*, with the former receiving overwhelmingly negative, the latter receiving overwhelmingly positive reviews, and Starbucks receiving mixed reviews consistent with the overall distribution of sentiment in the data set.



(a) *Top category* (b) *Name*

FIGURE 11.16: *Histograms illustrating the relationship between sentiment and (a) the top category and (b) the name of the associated establishment*

Bubble maps were employed as the type-specific visualisation of location information in the ECCO system. Whilst the bubble map view of the SMS data in Figure 9.39 did not reveal much insight into the data, given the skewed class distribution and spread of data points consistent with the population density of the country, the map view of the Yelp data set shows a distinguishable concentration of data points in the South West of the United States, with a few data points scattered across the north east of the country and southern Canada, as well as in North Carolina, as shown in Figure 11.17.

This insight was reinforced in the histogram of Figure 11.18(a), illustrating the distribution of sentiment according to state. From this figure, it is clear that most of the reviews in the Yelp data set (which were sampled randomly from the larger data set) were on establishments located in Nevada or Arizona, with the proportion of negative reviews exhibited in Arizona significantly larger than that of Nevada. By zooming in on areas with many reviews, the largest concentrated area of reviews was identified as the Las Vegas strip, shown in Figure 11.18(b), with *Gordon Ramsay Hell's Kitchen* receiving the largest number of reviews for an individual establishment. Compared to the establishment to its left (*Bacchanal Buffet*), the restaurant received a significantly larger proportion of positive reviews. Although McDonald's and Starbucks were shown to have received the largest number of reviews in Figure 11.16, the bubble map view revealed that these reviews were, in fact, distributed over several franchises across the United States and southern Canada.

The final type-specific visualisation implemented in the ECCO system is the time series graph representing the number of reviews submitted in each sentiment class per day. In Figure 9.43, it was shown that a general increase in reviews was exhibited towards the end of the year for the

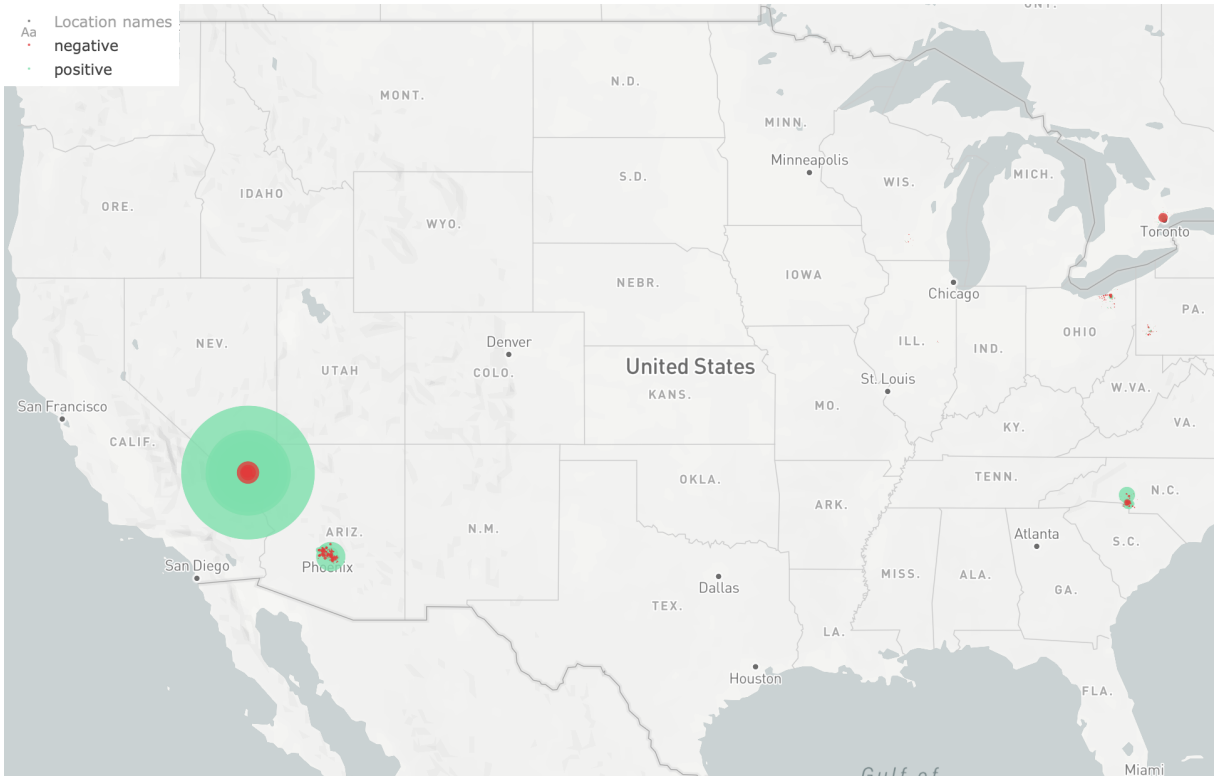
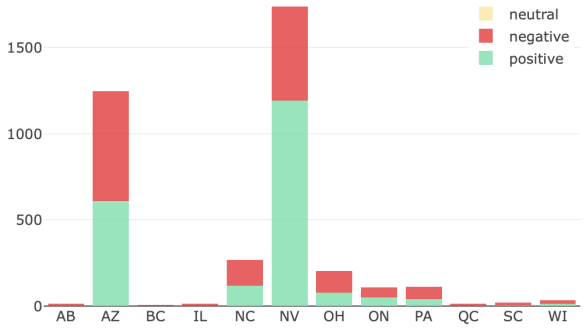
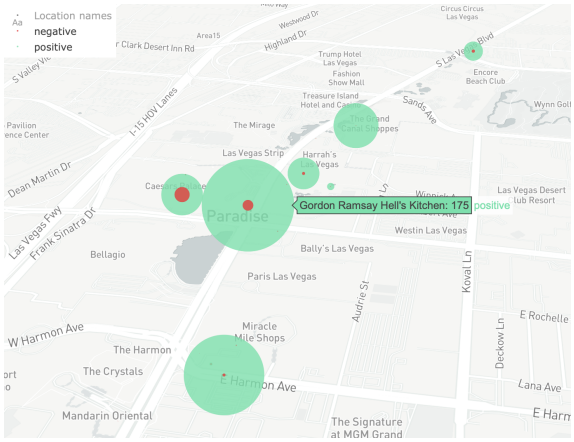


FIGURE 11.17: *A bubble map representation of the Yelp case study data elucidating the distribution of sentiment over the United States.*



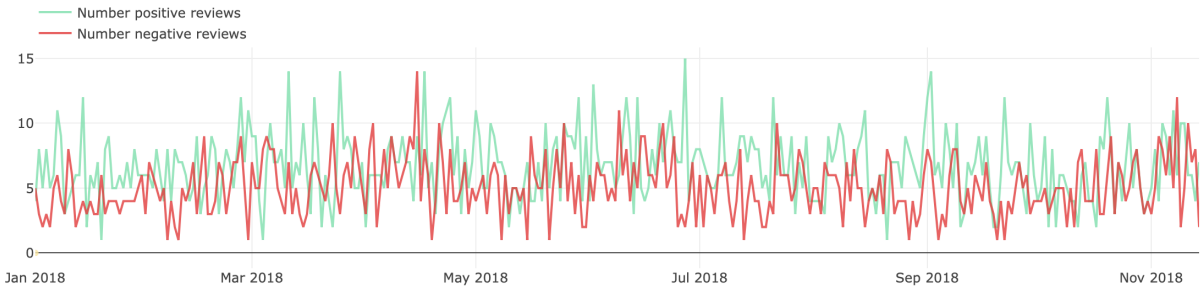
(a) *Sentiment distribution per state*



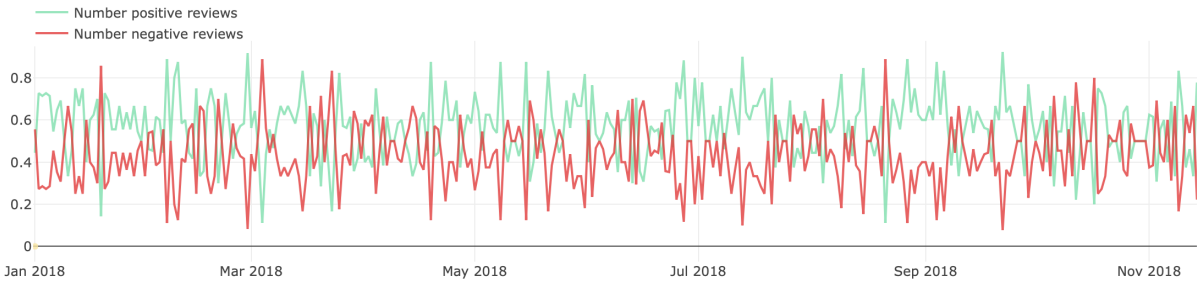
(b) *Map view of sentiment per establishment*

FIGURE 11.18: *A visualisation of (a) the relationship between sentiment and state, and (b) a zoomed-in version of the bubble map in Figure 11.17 for the Yelp data set.*

SMS data set. As shown in Figure 11.19(a), the number of reviews in the reduced Yelp data set is approximately constant throughout the year 2018. The normalised counts in Figure 11.19(b) reveal that the number of positive reviews typically exceeds that of negative reviews submitted daily, and this is particularly prevalent around late Januray/early February, March, July and late August/early September. In late February, April, June, August and October/November, on the other hand, the proportion of negative reviews increases to match that of positive reviews. Overall, however, a prominent pattern is not immediately evident.



(a) *Raw counts*



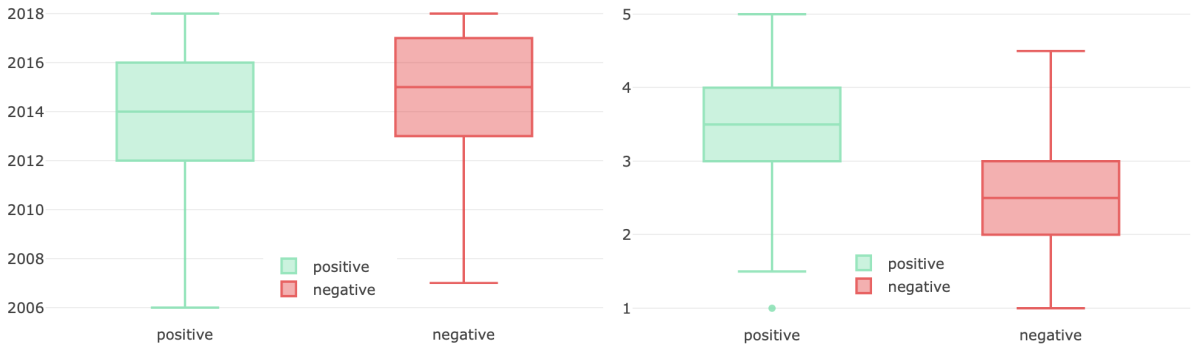
(b) *Normalised counts*

FIGURE 11.19: *Sentiment counts by date for the Yelp data set in (a) raw and (b) normalised format.*

With the use of the filtering function of Process 14.0, the visualisations described above can be made increasingly fine-grained. By filtering the time series graph in Figure 9.43 according to keyword, for example, the increase in reviews towards the end of the year was isolated to reviews containing the keyword *loan* in the SMS data set. Similarly, by analysing customer ratings for separate keywords in Figure 9.38, the relative importance of concerns for customers was identified. In respect of the Yelp data set, filtering on they keyword *location* produced slightly different results than in the general case for the joining year of the associated user and average stars of the associated establishments, as shown in Figures 11.20(a) and 11.20(b), respectively. More specifically, reviews containing the keyword *location* were associated with users who joined the site earlier, particularly in the case of positive reviews, as well as with establishments who had lower average star ratings compared to the general case. Location therefore appears to be an attribute more commonly discussed and favoured by longer-standing members of the site, whilst establishments whose location is a talking point in reviews typically fare worse in respect of average customer ratings.

Finally, latent relationships between structured variables and the sentiment class can be investigated by means of a multivariate model, as outlined in Process 17.0 of Figure 6.8. The classification tree illustrated in Figure 9.45 was, for example, fitted on the subset of reviews from the SMS data set which contain the keyword *service*, within which sentiment was approximately equally distributed between the positive and negative classes. In the case of the Yelp review data, a decision tree could be fit on the entire data set, since the class distribution is almost perfectly balanced.

The resulting tree, shown in Figure 11.21, achieved a test accuracy of 98.7% with the standard hyperparameters embedded into the ECCO system (as described in §7.2.3). The top split of the tree is the average star rating given by a user, whereby users who have an average star rating of at most 3.365 are predicted to submit negative reviews. Users with an average star rating greater than 3.365 are predicted to submit positive reviews, but the model is *more certain* of this



(a) *Yelping since year* (b) *Average stars establishment*

FIGURE 11.20: *The effect of filtering by keywords on the visualisations produced by the ECCO system. The graphs of (a) Figure 11.14(a) and (b) Figure 11.15(b) are shown, filtered on the keyword location.*

prediction if the average star rating of the establishment is greater than 3.75, and accordingly *less certain* if the average star rating is at most 3.75. In other words, based only on the average star ratings of the user and establishment associated with a review, the classification tree model can almost perfectly predict whether the review is positive or negative in sentiment. This further corroborates the findings discussed with respect to Figure 11.13.

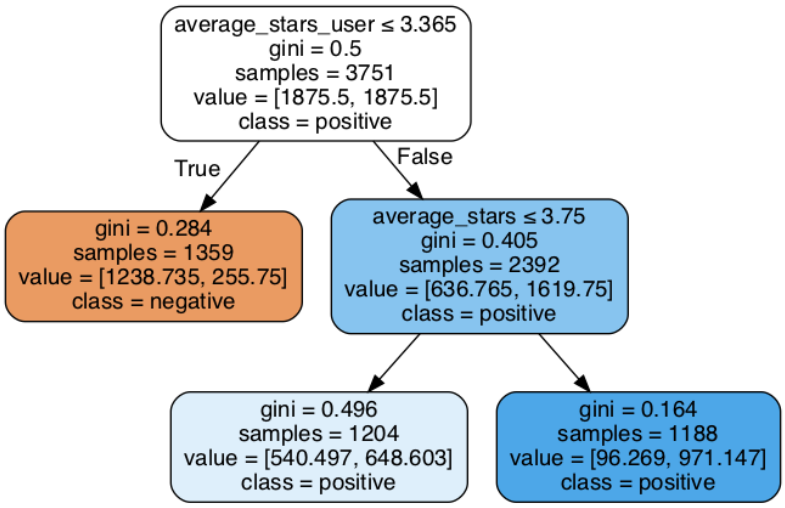


FIGURE 11.21: *A decision tree fitted to the Yelp data set, using the Gini impurity index subject to a minimum proportion of 10% of observations in order to warrant the split of a node and a minimum proportion of 30% of observations to be present in any given leaf node.*

As was illustrated in the examples above, the ECCO framework provides a unique approach towards analysing and sythesising sentiment information that does not rely on assumptions about the type of data being analysed (as many other frameworks do by analysing sentiment according to *product* features) and which facilitates an important and novel '*third layer*' of analysis by incorporating structured variables. This is achieved in an original manner by taking a *decision support* approach to data mining rather than compiling reports *automatically* according to some predefined pattern.

11.4 Discussion and comparison with other frameworks

The ECCO framework seeks to address two major shortcomings of existing frameworks in the literature, namely the non-existence of a framework that is *comprehensive* enough to be readily applied by analysts, yet *generic* enough to be applicable across various domains, as well as the lack of depth and flexibility afforded by existing frameworks to analyse model results with a view to inform decision making in response to sentiment information.

The framework combines the preprocessing, sentiment modelling and analysis steps of a sentiment analysis problem into one comprehensive framework that is generally applicable. The lack of such a framework in the literature was elucidated in §6.1. The TOM framework [152], for example, combined preprocessing and modelling activities, but these steps are tailored specifically to the modelling of sentiment in Twitter data and the analysis of the results is not addressed. The Opinion Observer [177], on the other hand, focused on the analysis and synthesis of product reviews, but did not incorporate sentiment preprocessing or sentiment modelling activities.

In §11.1, it was shown that the preprocessing approach prescribed by the ECCO framework was applicable across four distinct domains. The introduction of user feedback and an iterative approach, moreover, facilitated the selection of a good preprocessing configuration in each particular case and provided initial insight into the data.

To the best of the author's knowledge, the ECCO framework is the first to incorporate the entire MST selection process for modelling sentiment by means of machine learning into a sentiment analysis framework, thereby affording a user the ability to construct robust sentiment models regardless of domain and problem type. In their paper on the Heracles framework, reviewed in §6.1, Schouten *et al.* [275] also highlighted the importance of model evaluation and comparison in text mining applications. In their framework, however, the target of this evaluation was the machine learning models themselves, as opposed to the MSTs (incorporating model-feature pairs and their hyperparameters) addressed in the ECCO framework. Feature engineering and hyperparameter tuning efforts are left to automated functions in external packages in the Heracles framework, whose suitability and performance were not evaluated. In §11.2, it was illustrated that, by following the process outlined in the ECCO framework, machine learning models could be developed that achieved performance competitive with benchmark results across four distinct domains in both binary and ternary problem classes, and that these models generally outperformed specific pre-trained lexicon-based models by a significant margin. Moreover, it was shown that the proposed complementary selection approach (combining several models in an ensemble) could further improve on these results.

The ECCO framework is also the first to explicitly explore the relationship between sentiment and supplementary structured variables, facilitating a deeper analysis of sentiment data and allowing decision makers to extract actionable insight from data in order to drive organisational improvement. Furthermore, the exploratory approach adopted in the analysis component allows for the analysis of data from various domains without prerequisite knowledge. Illustrative examples highlighted in §11.3 demonstrated the ability of the modules in the analysis component to extract valuable insights from all data sets in the validation suite, both with and without the presence of structured supplementary data. Common points of critique in movie reviews, prominent themes in a set of microblogs, biases inherent in business reviews, as well as the grievances of bank customers and potential causes thereof could all be analysed by means of the same structured, exploratory process outlined in the ECCO framework.

In comparison, the popular Opinion Observer framework [177] described in §6.1, for example, facilitates the analysis of sentiment data for product reviews by segmenting sentiment according

to product features, whilst the BESAHOT framework [143] by Kasper and Vela (also described in §6.1) facilitates the analysis of hotel reviews by displaying reviews related to various topics discussed by guests (*e.g.* room cleanliness) and summary statistics on overall reviewer demographics. The ECCO framework is applicable to both domains, and the summaries generated by each of the existing frameworks are implicitly incorporated into the framework by means of the histograms for qualitative data comparison (akin to the product feature-sentiment graphs of the Opinion Observer), the topic modelling functionality paired with the filtering function (akin to the summary by topic of BESAHOT) and the type-specific visualisations of supplementary data (akin to the summary statistics of reviewer demographics of BESAHOT). Furthermore, the ECCO framework would also facilitate an investigation of the relationship between sentiment and supplementary data describing guests (*e.g.* do single business travellers complain more than families?), which is not supported by the BESAHOT framework. The ECCO framework therefore offers both versatility and depth of analysis.

11.5 Chapter summary

The focus of this chapter was the validation of the ECCO framework. The general applicability and utility of the framework was demonstrated by applying the ECCO system, an instantiation of the ECCO framework, to data sets from four distinct domains. During this process, the framework was shown to be effective in finding suitable preprocessing configurations, developing accurate sentiment models and extracting valuable insights across all considered domains. The models developed by means of the framework were, furthermore, shown to be competitive with benchmark results in the three cases where this was possible. In a concluding discussion of the results of the validation analysis, the framework was compared with several existing frameworks from the literature in order to further verify that the framework's objectives aimed at addressing various shortcomings in the literature had been fulfilled.

Part V

Conclusion

CHAPTER 12

Dissertation summary and appraisal

Contents

The purpose of this chapter is twofold: First, to present a summary of the work contained in this dissertation and secondly to offer an appraisal of the contributions of the dissertation.

12.1 Dissertation summary

Apart from the introductory chapter, this dissertation contains a total of twelve further chapters, which have been organised into five parts. Part I was devoted to a review of the pertinent literature, in fulfilment of Objective I of §1.3. The first chapter in Part I, Chapter 2, served as a review of prerequisite mathematical and statistical material necessary for the understanding of the techniques employed later in the dissertation. An introduction to random variables and statistical distributions was first given, and this was followed by a description of the Bernoulli, binomial and multinomial (discrete) distributions, as well as the Gaussian distribution, and the beta and Dirichlet (continuous) distributions. Several concepts related to matrices were subsequently reviewed, namely the rank of a matrix, matrix norms, eigenvectors and eigenvalues, as well as the process of singular value decomposition. Three important statistical models for the compact representation of data, namely Prinicipal Component Analysis, Latent Semantic Analysis and Latent Dirichlet Allocation, were then described. Finally, non-linear optimisation problems were considered. In particular, three gradient-based optimisation algorithms for non-linear, unconstrained optimisation problems were introduced, namely gradient descent, Newton's method and the BFGS algorithm. Furthermore, algorithms were described for solving constrained, non-linear optimisation problems, namely the method of Lagrange multipliers (used to solve problems with equality constraints) and the KT conditions (used to solve problems with inequality constraints). The notion of a genetic algorithm was also reviewed, which may be employed to compute approximate solutions to complex optimisation problems.

Chapter 3 contained a survey of the literature related to machine learning, in fulfilment of Objective I(a). An introduction to the notion of machine learning was first given, and this was followed by a description of various types of learning that prevail in this field, namely supervised, unsupervised, semi-supervised, weakly supervised and reinforcement learning. The differences between classification and regression problems, generative and discriminative models, as well

as probabilistic and scoring classifiers was, furthermore, illustrated. A general procedure for training and evaluating machine learning models was subsequently outlined, before hyperparameter tuning, model generalisability and the bias-variance trade-off were described in more detail. Performance evaluation metrics, including accuracy, precision, recall, the F-measure and the AUC value were also discussed. Finally, several machine learning algorithms were described, including kNN, tree-based methods, SVMs, naïve Bayes classifiers, binary and multi-class logistic regression (maximum entropy), as well as ensemble learning methods and deep neural networks. With respect to ensemble learning, three popular such methods, namely bagging, boosting and stacking, were described in detail. This was followed by a brief review of ensemble pruning or ensemble selection methods. The discussion on deep learning included training and design considerations specific to this type of machine learning algorithm, as well as several types of neural network architectures, namely feedforward, convolutional and recurrent neural networks, as well as autoencoders.

The fundamentals of sentiment analysis were reviewed in Chapter 4. The first section of this chapter was devoted to a discussion on the processing of unstructured data, in fulfilment of Objective I(b). A definition of data and the various types and forms they can assume was given, and this was followed by an outline of the generic data science process. Data processing steps employed to normalise unstructured text data were then described, namely tokenisation, stemming and lemmatisation. Various approaches for representing unstructured text in the form of a structured vector were subsequently outlined. These included term-based, linguistic and topic-oriented features. Furthermore, dimensionality reduction and feature selection techniques were discussed, including dictionary size reduction, feature transformation and representation learning. Sentiment analysis of unstructured text data was considered in the second section of the chapter, in fulfilment of Objective I(c). An overview of sentiment analysis was first given in the form of a brief timeline of research in the field, as well as a a structured categorisation of the research into sentiment analysis tasks and levels of granularity pursued in the analysis. A popular taxonomy was then employed to guide a discussion of the prevailing approaches towards classifying document-level sentiment polarity. During this process, the taxonomy was revised to reflect the current state of the literature. The process of summarising the results of the analysis was, finally, considered by reviewing textual and visual summaries of sentiment data.

The final chapter of Part I, Chapter 5, contained a review of the design and development of DSSs, in fulfilment of Objective I(d). A general description of DSSs was first given according to their primary components, namely the database component, the model component, the user interface and the communications component. Each of these components was then discussed in detail, and general guidelines or design considerations were reviewed in each respective section. Subsequently, a distinction was made between various types of DSSs. These include data-driven, document-driven, model-driven, knowledge-driven and communications-driven DSSs. The development of DSSs, and ISs in general, was then discussed in terms of the SDLC. Three popular systems development methodologies based upon the SDLC were subsequently described, namely the waterfall methodology, the agile methodology and the object-oriented methodology. Finally, appropriate measures for assuring the quality of an IS were outlined, including a top-down design approach and appropriate documentation, as well as verification and validation techniques.

Part II of this dissertation was dedicated to the presentation of a newly proposed sentiment analysis framework. In Chapter 6, the design of this generic framework for evaluating a corpus characterised by opinion-bearing language, the ECCO framework, was put forward, in fulfilment of Objective II. In order to place the proposed framework within the context of the current literature, an overview of similar existing frameworks was first given. The shortcomings of these frameworks were highlighted, which included a lack of generalisability due to full automation,

a focus on the deployment of a single, specific model rather than the facilitation of custom model development, and a lack of flexibility when analysing model results in order to extract actionable insights. Subsequently, a generic data science paradigm was proposed within which the ECCO framework was developed. In this manner, the generic nature of the framework was ensured, as well as that it contains all the necessary components for extracting insight from raw data. Finally, the ECCO framework itself was described. To this end, a high-level overview of the framework was first given. Each of its three primary functional components, namely a processing component, a sentiment modelling component and an analysis component, were then described in detail and illustrated by means of DFDs.

In Chapter 7, the ECCO system was demonstrated: A proof of concept implementation of the ECCO framework developed in the `Python` programming language, in pursuit of Objective III. The chapter opened with an account of the technical implementation of the system, including its object-oriented design and the libraries and frameworks that were used to develop it. Subsequently, the ECCO system was demonstrated in respect of the preprocessing and sentiment modelling stages facilitated primarily by the *MainWindow* interface, as well as the analysis stage primarily facilitated by the *Dashboard* interface. During this demonstration, the implementation of each of these stages was described in detail with reference to the algorithms and parameters selected to populate the ECCO framework, and illustrated by means of screen shots of the actual system. Finally, the measures taken to assure the quality of the system (verification and validation) were outlined, in fulfilment of Objective IV.

In Part III of this dissertation, a real-world case study was conducted in order to demonstrate the value of the ECCO framework, in fulfulment of Objective V. A background to this case study, which relates to a South African retail bank and feedback received from its customers, was provided in Chapter 8. The data collection and data preparation processes were also described in this chapter, including the generation of a *ground truth* annotated subset of the data and the merging of various data sources. Finally, the objectives pursued during the case study analysis were delineated. The results of this analysis were presented in Chapter 9. More specifically, the processing steps performed on the data were first described, illustrating the extent to which each of the filtering and normalisation steps contributed to the reduction of the vocabulary size. The final resulting vocabulary was over 60% smaller than the original, unprocessed vocabulary. The model development process was subsequently delineated, including the approach taken to tune various hyperparameters and the results of the comparative evaluation of the models generated by the ECCO system and various ensembles formed by subsets of these models, as well as the model employed by a third-party vendor contracted by the retail bank. Finally, the results of the selected CNN model were analysed by means of the *Dashboard* interface. During this analysis, the contents of the customer reviews were revealed and the importance of various topics to customers was elucidated. Furthermore, relationships between branch and customer attributes and the complaints related to various topics were explored. It was concluded that custom models generated by means of the ECCO system were able to outperform off-the-shelf models significantly, and that the Dashboard interface could be used effectively to gain insight into the grievances of customers and their possible causes.

After having demonstrated the practical applicability of the framework in detail in Part III, the focus shifted in Part IV to the validation of the framework, in fulfilment of Objective VI. More specifically, the framework's ability to generalise across domains was evaluated, as was its ability to achieve consistent classification performances competitive with those achieved by other researchers in respect of published benchmark data. The benchmark data sets selected for this purpose, namely the PL04 data set, the Yelp data set and the Twitter data set, were presented in Chapter 10. Along with the SMS data set employed in Part III of the dissertation, these data

sets formed the validation suite in respect of which the ECCO framework was validated. The processes followed to prepare each of the benchmark data sets for input to the ECCO system were then described. Finally, the versatility and suitability of the validation suite were discussed. It was concluded that the variety in domain, medium and language usage across the validation data sets was sufficient to assess the generality of the framework.

The second and last chapter of Part IV, Chapter 11, contained the validation analyses conducted in respect of the validation suite. During the validation process, it was demonstrated that the ECCO framework was effective in finding suitable preprocessing configurations, developing accurate sentiment models and extracting valuable insights across all four domains considered. In respect of each of the three benchmark data sets, it was, furthermore, shown that the models developed by means of the framework were competitive with the best results published by other researchers. Finally, the framework was compared with several existing frameworks from the literature based on the results of the validation studies. During this comparison, it was confirmed that the framework's objective to address various shortcomings in the literature had indeed been fulfilled.

12.2 Appraisal of dissertation contributions

The main contributions of this dissertation are eight-fold. This section contains a brief summary and appraisal of each of these contributions.

Contribution I *A revised taxonomy of sentiment analysis techniques reflecting the current state of the literature*

> With a research body comprising over 7 000 papers, of which 99% have been published after 2004 [186], sentiment analysis may be described as a particularly young and active field. As a result of this rapid growth, work dedicated to categorising and organising techniques in the field is rather lacking. Chapter 4 of this dissertation contains an effort towards such an organisation, by first placing sentiment analysis in the greater context of unstructured data analysis and then categorising the research in the field according to analysis objectives (*tasks*), levels of granularity and analysis approaches. During this process, one of only a few available taxonomies of sentiment analysis approaches was critically evaluated and revised to reflect the current state of the literature.

Contribution II *The proposal of a generic paradigm for facilitating the data scientific process*

> Before presenting the ECCO framework in Chapter 6, a generic data science paradigm was first proposed within which the ECCO framework was developed. This paradigm integrates the data science process proposed by O'Neill and Schutt [222] with the widely adopted architectural guidelines for a DSS, and reflects the stages into which a model-driven DSS is typically partitioned (the formulation stage, the solution stage and the analysis stage). Similarities between the proposed paradigm and general frameworks for text mining were also illustrated. Although this paradigm is simple, it may provide a good starting point for the development of future frameworks aimed at facilitating the transformation of raw data into information by adopting a decision support approach. Furthermore, it may provide a structure for comparing, as well as a shared terminology for describing, similar frameworks.

Contribution III *The proposal of a generic framework for employing sentiment analysis to extract insight from unstructured text data*

The sentiment analysis framework proposed in Chapter 6 addresses the shortcomings of existing frameworks identified in the literature in three primary ways. First, the framework was designed to be generic in nature. This was achieved, on the one hand, by developing the framework within the generic data science paradigm mentioned in Contribution II above and, on the other hand, adopting a modular approach towards framework design. Any modules incorporated into the framework may thus be exchanged, modified or deleted in accordance with new findings in the literature and the objectives of the analysis in question without disrupting the correct functioning of other modules of the framework. A lack of generality in existing frameworks may be attributed to the objective of these frameworks to produce fully automated sentiment analysis systems. This problem is circumvented in the ECCO framework by adopting a decision support approach in which the evaluation of opinionated data is *facilitated* for a particular user rather than *automated*.

Secondly, whereas most existing frameworks in the literature implement a single, specific model, the user is guided through the development of a custom model in the ECCO framework. This approach was adopted in view of the fact that the effectiveness of models for sentiment classification depends on the domain in which they are applied and, in the case of machine learning models, on the features employed as input to the model, as well as on the values of (hyper)parameters of the model. To the author's best knowledge, the ECCO framework is the first to incorporate the process of selecting an appropriate MST into a framework for sentiment analysis. This is a valuable addition since MST selection can have a significant impact on model performance and since, according to Kumar *et al.* [165], it is the most time-consuming activity in any process that makes use of machine learning. By allowing the user to develop and comparatively evaluate several model-feature combinations during a single iteration, the *steering* step of the model selection process (described in §6.3.2) is rendered more efficient. Furthermore, by transferring the computational load required for parameter estimation and hyperparameter tuning from the user to the computer, the *execution* step is also simplified. Finally, the consumption step is enhanced by means of the comparative summaries and visualisations of previously tested models issued to the user. There is, however, still room for improvement. It is envisioned that future research following on the work presented in this dissertation may be focused on enhancing the implementation of the model selection process towards an efficient integration of machine learning into DSS design. Proposals for such improvements are given in the following chapter.

Thirdly, the ECCO framework facilitates a particularly flexible exploratory analysis of model results. This includes traditional summaries and topic analyses, as well as the investigation of relationships between sentiment and attributes from additional, structured data sources by means of visualisations specific to certain data types and by means of learning models for which the learnt associations or '*rules*' may be extracted and interpreted. Furthermore, the analysis component of the framework is designed in such a way that results from one approach may be employed to guide the analysis by means of another approach. By actively involving the user in the analysis stage and by offering a variety of different forms of sentiment summary, the process of information extraction is facilitated for a diverse range of problem settings and at various levels, from the surface-level analysis of frequently occurring terms to the discovery of latent relationships between data attributes.

The general applicability of the framework was validated in respect of four data sets over distinct domains, in different contexts and with a variety of meta-features in Chapter 11 of this dissertation. Furthermore, the framework was shown to address and successfully overcome shortcomings of existing frameworks in the literature. The ECCO framework was published in the reputable journal Decision Support Systems [145].

Contribution IV *A concept demonstrator implementation of the proposed sentiment analysis framework in the form of a computer program*

The design of the aforementioned sentiment analysis framework was not limited to a conceptual level only. In Chapter 7, a practical implementation of the framework in the form of a computer program was demonstrated. During this demonstration, details were given to describe how the framework may be realised, including a possible system structure in the form of a class diagram and the description of various algorithms and libraries employed to implement the modules of the framework, such as a spell correction algorithm and the `Tensorboard` visualisation library used to facilitate manual hyperparameter tuning. In this manner, the practical usability of the ECCO framework was illustrated. The computerised concept demonstrator implementation of the framework was, furthermore, verified and validated according to the generally accepted guidelines reviewed in §5.4.1 and §5.4.2, further corroborating the value of the framework.

Contribution V *The application of the proposed sentiment analysis framework to a real-world case study*

The utility of the ECCO framework was illustrated in the context of a real-world setting by means of the case study performed in Chapters 8 and 9. The computerised concept demonstrator implementation of the framework was used to analyse reviews from customers of a South African retail bank. These reviews exhibited poor grammatical structure, frequent misspellings and colloquialisms, as well as a mixture of various languages. Furthermore, due to the process by which the data were collected, a strong bias towards the negative sentiment class was present in the data. In spite of these challenges, the models generated according to the process outlined in the ECCO framework achieved median AUC scores of up to 0.9 over ten replications. Furthermore, the MSTs selected for the machine learning approach significantly outperformed commercial tools and existing lexicon-based models from the literature in terms of both the AUC score and the classification accuracy. Finally, the analysis tools contained within the *Dashboard* and *LDAvis* interfaces of the framework implementation were successfully employed to extract information from the data, including common targets of the grievances expressed by customers and their relative importance, as well as specific reasons for the dissatisfaction associated with these targets, and customer and branch characteristics commonly associated with particular problems.

The results of this case study were summarised in a second journal article [146], which has been accepted for publication in the journal ORiON of the Operations Research Society of South Africa.

Contribution VI *The application of the proposed sentiment analysis framework to benchmark data sets from various domains*

The focus of Chapter 11 was to validate the ability of the ECCO framework to generalise across domains. To this end, the framework was successfully applied to four data sets from distinct domains with different contexts and meta-features. The validation suite included the data set from the banking industry employed in Chapters 8 and 9, as well as three benchmark data sets from the literature, related to film reviews, business reviews

and microblogs from a social media site. The framework was shown to be effective in uncovering suitable preprocessing configurations, developing accurate sentiment models and extracting valuable insights across all considered domains. The models developed by means of the framework were, furthermore, shown to be competitive with benchmark results in the three cases where this was possible.

The results of the validation analyses in respect of the benchmark data sets were included in the journal paper in which the ECCO framework itself was described [145].

Contribution VII *The proposal of a novel ensemble selection technique*

As discussed in §3.4.5, the problem of selecting suitable base learners for an ensemble model from a pool of candidates is under-explored in the literature. Many existing techniques involve comparing individual prediction errors of candidate models in respect of training or validation data in order to select a diverse and accurate set of base learners. As the number of candidate models increases, however, these approaches become expensive in terms of storage and computing power, since all candidate models must be trained in respect of the same data and their individual predictions stored. In Chapter 9 of this dissertation, a different approach was proposed, in which a set of base learners is selected by maximising a novel *favourability* metric — a weighted sum of measures of ensemble diversity, accuracy and size. The proposed diversity measure is based on meta-features of the models, rather than their individual predictions. In this dissertation, for instance, ensemble diversity was quantified as the degree of differentiation between base learners in respect of the learning algorithm and feature set employed. This measure of diversity can be computed prior to deploying candidate models and does not require individual predictions to be stored. If some measure of accuracy is already available for candidate base learners, such as historical performance in respect of similar tasks, the proposed ensemble selection approach, furthermore, avoids the costly re-training of candidate models that are not ultimately selected.

This approach was applied both in the context of the case study in Chapter 9 and the validation studies in Chapter 11. Overall, promising results were observed. The ensembles formed by means of the proposed approach slightly outperformed those associated with a greedy selection approach in two out of the four cases evaluated. More interestingly, an ensemble of the base learners selected by means of the favourability model achieved a 1% improvement in median accuracy over the best individual model in respect of the SMS data set, in spite of the average base learner accuracy being 14% lower than that of the individual model. The selection of a diverse set of base learners according to the proposed diversity metric therefore appears to provide some advantage. There is, however, considerable room for improvement of this approach, as is discussed in the following chapter in the context of possible future work.

Contribution VIII *The application and relative comparison of various ensemble configurations in respect of data sets from various domains*

In spite of a growing interest in ensemble learning techniques in the machine learning community, the application of ensemble models to sentiment classification is still limited, as discussed in §3.4. The application and comparative evaluation of several different ensemble learning techniques to benchmark data sets from the sentiment analysis literature, presented in Chapters 9 and 11, may therefore be viewed as a valuable contribution of this dissertation.

Six different ensemble configurations (the combination of discrete and scoring model outputs by means of simple voting, weighted voting and meta-learning, respectively)

were employed in conjunction with three different ensemble selection methods (a greedy approach and two variants of the proposed favourability model of Contribution VII) in respect of all four data sets in the validation suite.

Median performance improvements of up to 5.53% over the best individual model were observed for several ensemble configurations in respect of all four validation data sets. Furthermore, clear trends were identified that may prove useful to other researchers in the field. More specifically, the combination of scoring or probabilistic outputs by means of meta-learning matched or outperformed the remaining five ensemble configurations in respect of all the data sets in the validation suite. Moreover, in nine out of twelve cases ensembles employing scoring outputs outperformed those utilising discrete outputs.

A third paper comparing the effectiveness of various ensemble configurations in the context of sentiment analysis is being prepared for submission.

CHAPTER 13

Suggestions for future work

A number of suggestions for possible future follow-up work, building upon the work presented in this dissertation, are provided in this final chapter. These suggestions are partitioned into three sections, related to the proposed sentiment analysis framework, the implementation of this framework and the application of the framework to case study data, respectively. In each case, the suggestion is stated formally and then elaborated upon and motivated briefly.

13.1 Suggestions related to the proposed framework

This section contains suggestions for further work related to the proposed framework, the ECCO framework, presented in Chapter 6 of this dissertation.

Proposal I *Enlarging the scope of the framework to include aspect-level sentiment analysis*

The framework was limited in scope to perform sentiment classification at the document level. More specifically, the sentiment polarity of a document was determined by means of the modules of the *sentiment modelling* component of the framework, and these results were later summarised in respect of automatically extracted aspects (or *keywords*) using the modules of the *analysis* component of the framework. An alternative approach would be to first extract aspects from each review and then determine the sentiment polarity of text excerpts related to a given aspect, as described in §4.2.2. Adopting this approach would result in more accurate results, especially in cases where several differing sentiment polarities are expressed in the same document (*e.g. "the service was great, but the food was terrible"*). This may, however, also require such phrases to be labelled individually or necessitate the use of weakly supervised models (described in §3.1). Furthermore, the algorithm employed to perform aspect extraction may need to be refined. The algorithm applied in the proof of concept implementation of the framework in Chapter 7 makes use of parse trees to identify noun phrases. Due to its simplicity, this algorithm can produce imperfect results, and therefore serves merely as a guide for a human analyst, who makes the final selection. More sophisticated models for aspect extraction include *association rule mining* and attention-based recurrent neural networks, which learn to emphasise aspect-related terms during the training of aspect embeddings [358].

Proposal II *Including lower-level descriptions for selected processes*

One of the desired characteristics of the proposed framework is that it is generic in nature. Unlike many other frameworks in the literature, the ECCO framework therefore

does not include low-level descriptions of algorithm-specific processes. Typically, existing algorithms may easily be implemented to execute these processes by means of existing software libraries or frameworks. In the interest of practical applicability, however, it may be beneficial to include lower-level abstractions of processes related to certain popular algorithm *types* in cases where these processes are complex or not typically packaged in a software library. A lower-level abstraction may, for example, be provided for the training of semi-supervised learning models (a special case of Process 8.1 of the ECCO framework) according to various algorithm types (*e.g.* wrapper methods, graph-based methods and methods based on the autoencoder network, as described in §4.2.3). Similarly, a lower-level description of corpus-based and dictionary-based approaches to lexicon generation (Process 10 of the ECCO framework), various types of search algorithms for hyperparameter tuning (Process 8.2 of the ECCO framework) or the combination of various models to form ensembles (Process 10 of the ECCO framework) may be provided in order to provide further guidance to analysts wishing to apply the framework in a practical environment.

Proposal III *Tailoring the framework to a particular problem domain*

Sentiment analysis is a field with vast applications. As mentioned in the introductory chapter, measurements of public sentiment online have, for example, been shown to correlate with shifts in the stock market [19], the results of political elections and violent uprisings [176, 325]. The framework presented in this dissertation was designed to be generic in nature and adaptable to various problem settings. A possible avenue of future work may thus be to develop a specific instance of the framework tailored to address a particular problem domain. Modules in the analysis component of the framework could, for example, be customised to facilitate a targeted analysis of text data from social media or financial blogs along with supplementary data from the stock market (*e.g.* the average share price or the number of trades made for a particular stock).

13.2 Suggestions related to the proof of concept implementation of the framework

In this section, natural extensions are presented for the proof of concept implementation, the ECCO system demonstrated in Chapter 7 of this dissertation.

Proposal IV *Including lexicon-based algorithms for building a domain-specific sentiment lexicon*

The current version of the ECCO system implements only pre-trained lexicon-based models for sentiment analysis. Most of these models make use of sentiment lexicons that were generated using general-purpose English dictionaries or lexical resources, such WordNet. One of these models, Vader, also included common expressions used in the context of social media. None of these sentiment lexicons were generated in the context of the South African banking sector, however. This poses the question whether a lexicon-based model making use of a sentiment lexicon generated for the specific context would achieve better performance in respect of the case study data of Chapters 8 and 9 than pre-trained models which faired poorly compared to machine learning models. Similar arguments could be made for the remaining case studies in the validation suite. The superior performance of the Vader model in respect of the Twitter data set, for which it was developed, further supports this hypothesis. In order to alleviate this problem, a sentiment lexicon may, for example, be generated by means of Turney's corpus-based SO-A algorithm [321] in respect of the corpus uploaded by the user.

Proposal V *Extending the set of available feature generation and feature selection techniques*

As described in §4.1.2, feature extraction methods for natural language go beyond the term-based features currently employed by the ECCO system to include linguistic features and topic-oriented features, as well as other surface-level features such as the length of a particular review. Although the currently implemented bag-of-n-grams model constitutes one of the most popular feature representations for text, it may be interesting to investigate the effects of other features on model performance. Furthermore, several methods for feature selection may also be applied, as described in §4.1.3. Currently, document frequency is employed as a means of feature selection to reduce the vocabulary size for all machine learning models, whilst word embeddings are trained end-to-end for the CNN and LSTM models only. Given the fact that word embeddings mitigate the problem of data sparsity in addition to reducing the feature space, it would be worthwhile to investigate whether other machine learning models exhibit an improvement in performance when word embeddings are used instead of the sparse term-document-matrix generated by the bag-of-n-grams model. These embeddings may be generated by transforming the term-document-matrix using PCA or LSA. Alternatively, pre-trained embeddings may be employed from publicly available resources, such as Word2Vec, or from the CNN and LSTM models. It is unclear, however, whether word embeddings generated by means of representation learning separate from the sentiment analysis task (*i.e.* Word2Vec) would be beneficial for classifying sentiment. Finally, a word embedding layer could be integrated with the other machine learning models as was done for CNN and LSTM. This would, however, complicate the loss function of the '*shallow*' models. Consequently, gradient-based optimisation algorithms may have to be employed where this is not already the case.

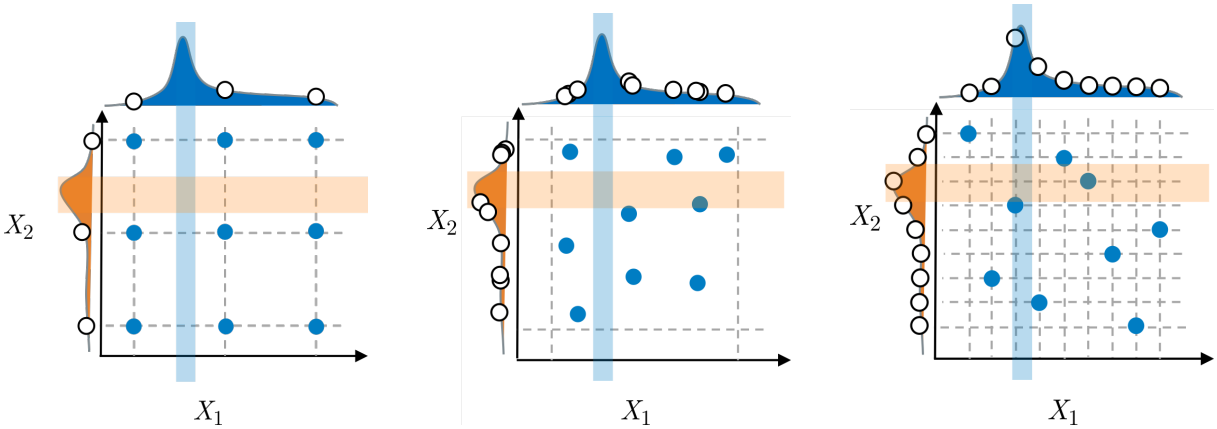
Proposal VI *Implementing semi-supervised machine learning algorithms*

One of the drawbacks of supervised machine learning algorithms is their reliance on annotated data, since data labels are rarely available and the process of manually labelling individual observations is time-consuming and expensive. This is especially true for low-resource languages, as may be present in data originating within a South African context, given the country's eleven official languages. Unlabelled data, on the other hand, are typically readily available. Machine learning algorithms which partially overcome this problem are semi-supervised learning algorithms (described in §3.1 and §4.2.3). These methods leverage both unlabelled and labelled data sources, in an effort to achieve similar performance whilst using fewer labelled observations. It is suggested that such an algorithm be implemented in the proof of concept demonstration in order to investigate to what extent such methods could reduce the computational load on organisations and analysts to label their data. For models which employ word (and document) embeddings, labels of annotated documents could, for example, be propagated to documents whose embedding vectors resemble those of the annotated documents in terms of some measure of matrix similarity.

Proposal VII *Improving the hyperparameter tuning process*

The ECCO framework is the first of its kind to integrate the selection of an MST into the sentiment analysis process. In this manner, the development of custom models is facilitated which achieve a desired level of performance in respect of data from the given domain. In the current implementation of the ECCO system, a grid search is employed to tune the hyperparameters of machine learning models based on a user-specified parameter grid. Whilst this approach is guaranteed to find the best solution from the given parameter grid, it is computationally expensive, since every possible combination

of hyperparameter values from the parameter grid must be evaluated by means of k-fold cross-validation. A grid search must therefore be relatively coarse in order to be feasible [163]. As shown in Figure 13.1(a), good combinations of hyperparameters may therefore be missed by such an algorithm. In order to alleviate this problem, a more efficient hyperparameter search algorithm could be applied.



(a) *Grid search* (b) *Random search* (c) *Latin hypercube sampling*

FIGURE 13.1: *Approaches to hyperparameter tuning. Hypothetical distributions of two hyperparmeters X_1 and X_2 with respect to a training objective are shown along with a nine-point search for (a) a grid search, (b) a random search and (c) Latin hypercube sampling (adapted from Koch et al. [163]).*

One such algorithm is the random search (as mentioned in §3.2.2), where a fixed number of hyperparameter combinations is evaluated, each comprising randomly selected hyperparameter values from a user-specified range. This approach, while less exact, has been shown to be empirically more efficient than a grid search [26]. This is, in part, due the larger number of values that may be explored for each hyperparameter at the same computational cost than in a grid search, as shown in Figure 13.1(a)–(b). A similar approach is to use random *Latin hypercube sampling* [189], where samples are uniform across each hyperparameter but random in combinations, filling the search space more efficiently, as shown in Figure 13.1(c) [163].

Finally, *intelligent* search algorithms could also be applied which typically employ a meta-heuristic or meta-*learning* approach towards find good hyperparameter values (as mentioned in §3.2.2). Such methods could be deployed as an *autotune* feature of the ECCO system which does not require any user input. Such an approach, if proven effective, would substantially reduce the load on the user of the system without compromising on the ability to select a suitable MST for each new problem instance.

Proposal VIII *Improving upon the functionality for ensemble selection*

In the current implementation of the ECCO system, the user is required to select base learners manually from the available models in order to form an ensemble. As discussed in §3.4.5, however, selecting suitable base learners is a complex task, especially as the number of candidate models becomes large. Implementing an automatic ensemble selection approach in the system would reduce the workload of the user considerably, and potentially improve the performance of the resulting ensemble models if an effective approach is employed. Possible ensemble selection approaches include search-based methods, clustering-based methods and ranking-based methods (see §3.4.5). The greedy search

and the proposed favourability model employed in Chapters 9 and 11 could, for example, be integrated into the ECCO system.

With respect to the proposed favourability model, several avenues for improvement can be pursued based on the observations made during the case studies. More specifically, although ensembles of base learners selected according to the proposed model were shown to leverage internal diversity successfully and thereby significantly improve on average base learner accuracy, computed ensemble favourability values did not correlate particularly well with relative ensemble performance. A different weighting of the composite metrics of diversity, accuracy and ensemble size may produce a more effective measure of ensemble favourability. Furthermore, it was observed that imposing a minimum threshold on base learner accuracy could improve the performance of the resulting ensemble models.

13.3 Suggestions related to case studies

This section contains suggestions related to the case study performed in Chapters 8 and 9, as well as the validation studies performed in Chapters 10 and 11.

Proposal IX *Expanding and diversifying the validation suite*

The data sets in the validation suite described in Chapter 10 were considerably diverse in domain, medium (SMS, social media, review sites), document length and vocabulary size. Due to limited computing power available to the author, however, only four data sets were selected, all of which are relatively small in size. In order to further strengthen the argument of generality of the ECCO framework, it would be valuable to apply it to several additional data sets, with a greater variation in size as well as the above-mentioned meta-features. Classifying the sentiment conveyed in news headlines and investigating the relationship of this sentiment with measurable reactions of readers, for example, is a possible additional use case of the ECCO framework.

Proposal X *Applying the framework to a different problem type*

The ECCO framework was presented and validated in this dissertation in the context of a sentiment classification problem. As mentioned in §6.3, however, the framework was constructed in such a manner that it may also be adapted to different problem settings. Instead of classifying sentiment conveyed in documents, the framework may, for example, be employed to classify documents into different categories (*e.g. spam*, work and personal e-mails, or news articles on various topics). The analysis component of the framework may then be used in a similar manner as it was employed during the case studies, in order to investigate word usage and typical profiles described by supplementary data with a view to extract valuable insights. Some minor adjustments may have to be made to the ECCO system in order to facilitate this new class of problems.

References

[1] Adamec Č & VIDEN I, 1947, *Polls come to Czechoslovakia*, Public Opinion Quarterly, **11(4)**, pp. 548–552.

[2] Ágel V, Eichinger LM, Eroms HW, Hellwig P, Heringer HJ & Lobin H (Eds), 2006, *Dependency and Valency: An International Handbook of Contemporary Research*, Walter de Gruyter, Berlin.

[3] Ajmera J, Ahn H-I, Nagarajan M, Verma A, Contractor D, Dill S & Denesuk M, 2013, *A CRM system for social media: Challenges and experiences*, Proceedings of the 22nd International World Wide Web Conference, Rio de Janeiro, pp. 49–58.

[4] Alter S, 1980, *Decision Support Systems: Current Practice and Continuing Challenges*, Addison-Wesley Publishing Company, Boston (MA).

[5] Ambler SW, 2013, *UML 2 class diagramming guidelines*, [Online], [Cited July 2019], Available from `http://www.agilemodeling.com/style/classDiagram.htm`.

[6] Annett M & Kondrak G, 2008, *A comparison of sentiment analysis techniques: Polarizing movie blogs*, Proceedings of the Conference of the Canadian Society for Computational Studies of Intelligence, Windsor, pp. 25–35.

[7] Anonymous, 2008, *Pearson's correlation coefficient*, pp. 1090–1091 in Kirch W (Ed), *Encyclopedia of Public Health*, Springer Netherlands, Dordrecht.

[8] Araque O, Corcuera-Platas I, Sánchez-Rada JF & Iglesias CA, 2017, *Enhancing deep learning sentiment analysis with ensemble techniques in social applications*, Expert Systems with Applications, **77**, pp. 236–246.

[9] Aronof M & Fudeman K, 2011, *What is Morphology?*, Wiley-Blackwell, Hoboken (NJ).

[10] Asghar MZ, Khan A, Ahmad S, Qasim M & Khan IA, 2017, *Lexicon-enhanced sentiment analysis framework using rule-based classification scheme*, PLoS ONE, **12(2)**, pp. 1–22.

[11] Asghar MZ, Khan A, Zahra SR, Ahmad S & Kundi FM, 2017, *Aspect-based opinion mining framework using heuristic patterns*, Cluster Computing, **doi: 10.1007/s10586-017-1096-9**, pp. 1–19.

[12] Atkinson K, 2016, *GNU Aspell*, [Online], [Cited May 2019], Available from `http://aspell.net/`.

[13] Bahdanau D, Cho K & Bengio Y, 2015, *Neural machine translation by jointly learning to align and translate*, Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015), San Diego (CA), pp. 1096–1103.

[14] Bai X, 2011, *Predicting consumer sentiments from online text*, Decision Support Systems, **50(4)**, pp. 732–742.

[15] Bainomugisha E, Carreton AL, van Cutsem T, Mostinckx S & de Meuter W, 2013, *A survey on reactive programming*, ACM Computing Surveys, **45(4)**, pp. 52:1–52:34.

[16] Balakrishnan N & Nevzorov VB, 2003, *A Primer on Statistical Distributions*, John Wiley & Sons, Hoboken (NJ).

[17] Bank M, 2013, *AIM — A social media monitoring system for quality engineering*, PhD Thesis, University of Leipzig, Leipzig.

[18] Banks J, Carson JS, Nelson BL & Nicol DM, 2010, *Discrete Event System Simulation*, 5th Edition, Pearson, Upper Saddle River (NJ).

[19] Bannister K, 2015, *Understanding sentiment analysis: What it is & why it's used*, [Online], [Cited February 2018], Available from `https://www.brandwatch.com/blog/understanding-sentiment-analysis/`.

[20] Basari ASH, Hussin B, Ananta IGP & Zeniarja J, 2013, *Opinion mining of movie review using hybrid method of support vector machine and particle swarm optimization*, Procedia Engineering, **53**, pp. 453–462.

[21] Baydin AG, Pearlmutter BA, Radul AA & Siskind JM, 2018, *Automatic differentiation in machine learning: A survey*, Journal of Marchine Learning Research, **18**, pp. 1–43.

[22] Bell R, 2009, *A beginner's guide to big O notation*, [Online], [Cited July 2019], Available from `https://rob-bell.net/2009/06/a-beginners-guide-to-big-o-notation/`.

[23] Bengio Y, 2009, *Learning deep architectures for AI*, Foundations and Trends in Machine Learning, **2(1)**, pp. 1–127.

[24] Bengio Y, Boulanger-Lewandowski N & Pascanu R, 2013, *Advances in optimizing recurrent networks*, Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, pp. 8624–8628.

[25] Bengio Y, Simard P & Frasconi P, 1994, *Learning long-term dependencies with gradient descent is difficult*, IEEE Transactions on Neural Networks, **5(2)**, pp. 157–166.

[26] Bergstra J & Bengio Y, 2012, *Random search for hyper-parameter optimization*, Journal of Machine Learning Research, **13**, pp. 281–305.

[27] Bespalov D, Bai B, Shokoufandeh A & Qi Y, 2011, *Sentiment classification based on supervised latent n-gram analysis*, Proceedings of the 20th ACM International Conference on Information and Knowledge Management, Glasgow, pp. 375–382.

[28] Bhuta S & Doshi U, 2014, *A review of techniques for sentiment analysis of Twitter data*, Proceedings of the International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT), Ghaziabad, pp. 583–591.

[29] Bird S, Klein E & Loper E, 2009, *Natural Language Processing with Python — Analyzing Text with the Natural Language Toolkit*, 1st Edition, O'Reilly Media, Sebastopol (CA).

[30] Bishop CM, 2006, *Pattern Recognition and Machine Learning*, Springer Science & Business Media, New York (NY).

[31] Blake C, 2006, *A comparison of document, sentence, and term event spaces*, Proceedings of the 44st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics, Sydney, pp. 601–608.

[32] Blake C, 2011, *Text mining*, Annual Review of Information Science and Technology, **45(1)**, pp. 121–155.

[33] Blei DM, Edu BB, Ng AY, Edu AS, Jordan MI & Edu JB, 2003, *Latent Dirichlet allocation*, Journal of Machine Learning Research, **3**, pp. 993–1022.

[34] Bonnans J-F, Gilbert JC, Lemaréchal C & Sagastizábal CA, 2006, *Numerical Optimization: Theoretical and Practical Aspects*, 2nd Edition, Springer Science & Business Media, New York (NY).

[35] Breiman L, 1996, *Bagging predictors*, Machine Learning, **24(2)**, pp. 123–140.

[36] Breiman L, Friedman JH, Olshen RA & Stone CJ, 1984, *Classification and Regression Trees*, Wadsworth International Group, Belmont (CA).

[37] Bridges J & Vásquez C, 2018, *If nearly all Airbnb reviews are positive, does that make them meaningless?*, Current Issues in Tourism, **21(18)**, pp. 2057–2075.

[38] BrightLocal, 2017, *Local consumer review survey*, [Online], [Cited February 2018], Available from `https://www.brightlocal.com/learn/local-consumer-review-survey/#Q11`.

[39] BrokenAlgorithm & Dustyatx, 2018, *Running calculation-heavy process in background?*, [Online], [Cited July 2019], Available from `https://community.plot.ly/t/running-calculation-heavy-process-in-background/17400/2`.

[40] Broyden CG, 1970, *The convergence of a class of double-rank minimization algorithms*, Journal of the Institute of Mathematics and its Applications, **6**, pp. 76–90.

[41] Bryan K, 2017, *Quasi-Newton methods*, [Lecture Notes], Department of Mathematics at the Rose-Hulman Institute of Technology, Terre Haute (IN).

[42] Burden RL & Faires JD, 1989, *Numerical Analysis*, 4th Edition, PWS Publishing Co., Boston (MA).

[43] Cambria E, Schuller B, Xia Y & Havasi C, 2013, *New avenues in opinion mining and sentiment analysis*, IEEE Intelligent Systems, **28(2)**, pp. 15–21.

[44] Cambria E, Poria S, Bisio F, Bajpai R & Chaturvedi I, 2015, *The CLSA model: A novel framework for concept-level sentiment analysis*, Proceedings of the International Conference on Intelligent Text Processing and Computational Linguistics, Mexico City, pp. 3–22.

[45] Carenini G, Cheung JCK & Pauls A, 2013, *Multi-document summarization of evaluative text*, Computational Intelligence, **29(4)**, pp. 545–576.

[46] Caruana R, Munson A & Niculescu-Mizil A, 2006, *Getting the most out of ensemble selection*, Proceedings of the International Conference on Data Mining, Hong Kong, pp. 828–833.

[47] Caruana R, Niculescu-Mizil A, Crew G & Ksikes A, 2004, *Ensemble selection from libraries of models*, Proceedings of the International conference on Machine learning, Banff, pp. 18–26.

[48] Chang C-C, Lin C-J & Tieleman T, 2008, *LIBSVM: A library for support vector machines*, ACM Transactions on Intelligent Systems and Technology (TIST), **307**, pp. 1–39.

[49] Chaovalit P & Zhou L, 2005, *Movie review mining: A comparison between supervised and unsupervised classification approaches*, Proceedings of the 38th Annual Hawaii International Conference on System Sciences, Big Island (HI), pp. 1–9.

[50] Chen T & Guestrin C, 2016, *Xgboost: A scalable tree boosting system*, Proceedings of the International Conference on Knowledge Discovery and Data Mining, San Francisco (CA), pp. 785–794.

[51] Chen Y, Wong M-L & Li H, 2014, *Applying ant colony optimization to configuring stacking ensembles for data mining*, Expert Systems with Applications, **41(6)**, pp. 2688–2702.

[52] Choi Y, Kim Y & Myaeng S-H, 2009, *Domain-specific sentiment analysis using contextual feature generation*, Proceedings of the 1st International CIKM Workshop on Topic-sentiment Analysis for Mass Opinion, Hong Kong, pp. 37–44.

[53] Chuang J, Manning CD & Heer J, 2012, *Termite: Visualization techniques for assessing textual topic models*, Proceedings of the International Working Conference on Advanced Visual Interfaces, Capri, pp. 74–77.

[54] Cieliebak M, 2018, *Sentiment analysis: Distinguish positive and negative documents*, [Online], [Cited July 2019], Available from `http://www.agilemodeling.com/style/classDiagram.htm`.

[55] Claesen M & de Moor B, 2015, *Hyperparameter search in machine learning*, Proceedings of the 11th Metaheuristics International Conference, Agadir, pp. 1–5.

[56] Clark S & Gales M, 2013, *Machine learning for language processing: Topic modelling and latent Dirichlet allocation*, [Lecture Notes], Department of Computer Science and Technology at the University of Cambridge, Cambridge.

[57] Coad P, Yourdon E & Coad P, 1991, *Object-Oriented Analysis*, Yourdon Press, Englewood Cliffs (NJ).

[58] Computational Linguistics & Psycholinguistics Research Center, 2018, *Pattern*, [Online], [Cited July 2019], Available from `https://www.clips.uantwerpen.be/pattern`.

[59] Cooijmans T, Ballas N, Laurent C, Gülçehre Ç & Courville A, 2017, *Recurrent batch normalization*, Proceedings of the 5th International Conference on Learning Representations, Toulon, [Poster session].

[60] Da Silva NFF, Coletta LFS & Hruschka ER, 2016, *A survey and comparative study of tweet sentiment analysis via semi-supervised learning*, ACM Computing Surveys, **49(1)**, pp. 1–26.

[61] Dale R, 2016, *The return of the chatbots*, Natural Language Engineering, **22(5)**, pp. 811–817.

[62] Das SR & Chen MY, 2007, *Yahoo! for Amazon: Sentiment extraction from small talk on the web*, Management Science, **53(9)**, pp. 1375–1388.

[63] Dave K, Lawrence S & Pennock DM, 2003, *Mining the peanut gallery: Opinion extraction and semantic classification of product reviews*, Proceedings of the 12th International World Wide Web Conference, Budapest, pp. 519–528.

[64] De Condorcet N, 1785, *Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix*, de l'Imprerie Royale, Paris.

[65] de Freitas, N, 2015, *Deep learning lecture 6: Optimisation*, [Lecture Notes], Department of Computer Science at the University of Oxford, Oxford.

[66] de Klerk E, Roos C & Terlaky T, 2004, *Nonlinear optimization (CO 367)*, [Lecture Notes], Faculty of Mathematics at the University of Waterloo, Waterloo.

[67] de Lucena CIM, 2016, *Framework for collaborative knowledge management in organizations*, PhD Thesis, NOVA University Lisbon, Lisbon.

[68] Deerwester S, Dumais ST, Furnas GW, Landauer TK & Harshman RA, 1990, *Indexing by latent semantic analysis*, Journal of the Association for Information Science and Technology, **41(6)**, pp. 391–407.

[69] Dennis A, Wixom BH & Roth RM, 2012, *Systems Analysis and Design*, 5th Edition, John Wiley & Sons, Hoboken (NJ).

[70] Devika MD, Sunitha C & Ganesh A, 2016, *Sentiment analysis: A comparative study on different approaches*, Procedia Computer Science, **87**, pp. 44–49.

[71] Dey L & Haque SM, 2009, *Opinion mining from noisy text data*, International Journal on Document Analysis and Recognition, **12(3)**, pp. 205–226.

[72] Dhaoui C, Webster CM & Tan LP, 2017, *Social media sentiment analysis: Lexicon versus machine learning*, Journal of Consumer Marketing, **34(6)**, pp. 480–488.

[73] Dhar V, 2013, *Data science and prediction*, Communications of the ACM, **56(12)**, pp. 64–73.

[74] Dietterich TG, 2000, *Ensemble methods in machine learning*, Proceedings of the International Workshop on Multiple Classifier Systems, Cagliari, pp. 1–15.

[75] Ding X, Liu B & Yu PS, 2008, *A holistic lexicon-based approach to opinion mining*, Proceedings of the 2008 International Conference on Web Search and Data Mining, Palo Alto (CA), pp. 231–240.

[76] Drakos G, 2018, *Handling missing values in machine learning: Part 1*, [Online], [Cited July 2019], Available from `https://towardsdatascience.com/handling-missing-values-in-machine-learning-part-1-dda69d4f88ca`.

[77] Ducange P, Fazzolari M, Petrocchi M & Vecchio M, 2019, *An effective decision support system for social media listening based on cross-source sentiment analysis models*, Engineering Applications of Artificial Intelligence, **78(2019)**, pp. 71–85.

[78] Dumais ST, 2004, *Latent semantic analysis*, Annual Review of Information Science and Technology, **38(1)**, pp. 188–230.

[79] Eisenhauer T, 2019, *Free and open source software (FOSS) versus paid software for your social business needs*, [Online], [Cited March 2012], Available from `https://axerosolutions.com/blogs/timeisenhauer/pulse/59/free-and-open-source-software-foss-vs-paid-software-for-your-social-business-needs`.

[80] el Sisi FN, Hegazy A E-F & Kader HA, 2015, *Opinion mining framework for news reviews to build good customer loyalty*, Proceedings of the 25th International Conference on Computer Theory and Applications, Alexandria, pp. 28–33.

[81] Ernst & Young, 2012, *Global consumer banking survey*, [Online], [Cited February 2018], Available from `http://www.ey.com/Publication/vwLUAssets/ey-global-consumer-banking-survey-2012/FILE/ey-global-consumer-banking-survey-2012.pdf`.

[82] Esuli A & Sebastiani F, 2006, *SentiWordNet: A publicly available lexical resource for opinion mining*, Proceedings of the 5th Conference on Language Resources and Evaluation, Genoa, pp. 417–422.

[83] Fan R-E, Chen P-H & Lin C-J, 2005, *Working set selection using second order information for training support vector machines*, Journal of Machine Learning Research, **6**, pp. 1889–1918.

[84] Fawcett T, 2006, *An introduction to ROC analysis*, Pattern Recognition Letters, **27(8)**, pp. 861–874.

[85] Fegiz PL, 1947, *Italian public opinion*, Public Opinion Quarterly, **11(1)**, pp. 92–96.

[86] Fletcher R, 1970, *A new approach to variable metric algorithms*, Computer Journal, **13(3)**, pp. 317–322.

[87] Fossbytes, 2016, *Most popular programming languages for machine learning and data science*, [Online], [Cited July 2019], Available from `https://fossbytes.com/popular-top-programming-languages-machine-learning-data-science/`.

[88] French CD, 1995, *"One size fits all" database architectures do not work for DSS*, Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose (CA), pp. 449–450.

[89] Freund Y & Schapire RE, 1995, *A desicion-theoretic generalization of on-line learning and an application to boosting*, Proceedings of the European Conference on Computational Learning Theory, Barcelona, pp. 23–37.

[90] Frith A, 2017, *Linguistic diversity map of South Africa*, [Online], [Cited July 2019], Available from `https://adrianfrith.com/linguistic-diversity`.

[91] Fuglede B & Topsoe F, 2004, *Jensen-Shannon divergence and Hilbert space embedding*, Proceedings of the International Symposium on Information Theory (ISIT), Chicago (IL), p. 31.

[92] Gal Y & Ghahramani Z, 2016, *Bayesian convolutional neural networks with Bernoulli approximate variational inference*, Proceedings of the 4th International Conference on Learning Representations (ICLR), San Juan.

[93] Gamon M, 2004, *Sentiment classification on customer feedback data: Noisy data, large feature vectors, and the role of linguistic analysis*, Proceedings of the 20th International Conference on Computational Linguistics, Geneva, pp. 841–847.

[94] Gamon M, Aue A, Corston-Oliver S & Ringger E, 2005, *Pulse: Mining customer opinions from free text*, Proceedings of the 6th International Symposium on Intelligent Data Analysis, Madrid, pp. 121–132.

[95] Gane C & Sarson T, 1979, *Structured Systems Analysis: Tools and Techniques*, Prentice-Hall, Englewood Cliffs (NJ).

[96] Ganesan K, 2013, *Opinion driven decision support system*, PhD Thesis, University of Illinois, Champaign (IL).

[97] Giacinto G, Roli F & Fumera G, 2000, *Design of effective multiple classifier systems by clustering of classifiers*, Proceedings of the International Conference on Pattern Recognition, Barcelona, pp. 160–163.

[98] Glasmachers T, 2017, *Limits of end-to-end learning*, Proceedings of the 9th Asian Conference on Machine Learning (ACML), Seoul, pp. 17–32.

[99] Glorot X & Bengio Y, 2010, *Understanding the difficulty of training deep feedforward neural networks*, Proceedings of the 13th International Conference on Artificial Intelligence and Statistics, Sardinia, pp. 249–256.

[100] Glorot X, Bordes A & Bengio Y, 2011, *Domain adaptation for large-scale sentiment classification: A deep learning approach*, Proceedings of the 28th International Conference on Machine Learning, Bellevue (WA), pp. 513–520.

[101] Goldfarb D, 1970, *A family of variable metric updates derived by variational means*, Mathematics of Computation, **24(109)**, pp. 23–26.

[102] Golub GH & van Loan CF, 1996, *Matrix Computations*, 3rd Edition, Johns Hopkins University Press, Baltimore (MD).

[103] Goodfellow I, Bengio Y & Courville A, 2016, *Deep Learning*, MIT Press, Cambridge (MA).

[104] Google, 2013, *word2vec [Code archive]*, [Online], [Cited August 2018], Available from `https://code.google.com/archive/p/word2vec/`.

[105] Grama A, Gupta A, Karypis G & Kumar V, 1994, *Introduction to Parallel Computing: Design and Analysis of Algorithms*, Addison-Wesley Professional, Boston (MA).

[106] Gregory ML, Chinchor N, Whitney P, Carter R, Hetzler E & Turner A, 2006, *User-directed sentiment analysis: Visualizing the affective content of documents*, Proceedings of the Workshop on Sentiment and Subjectivity in Text, Sydney, pp. 23–30.

[107] Grimes S, 2008, *Unstructured data and the 80 percent rule*, Clarabridge Bridgepoints Newsletter, August, pp. 1–2.

[108] Gu S, Cheng R & Jin Y, 2015, *Multi-objective ensemble generation*, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, **5(5)**, pp. 234–245.

[109] Guan Z, Chen L, Zhao W, Zheng Y, Tan S & Cai D, 2016, *Weakly-supervised deep learning for customer review sentiment classification*, Proceedings of the International Joint Conference on Artificial Intelligence, New York (NY), pp. 3719–3725.

[110] Gurusamy V & Kannan S, 2017, *Performance analysis: Stemming algorithm for the English language*, International Journal for Scientific Research and Development, **5(5)**, pp. 1933–1938.

[111] Hailong Z, Wenyan G & Bo J, 2014, *Machine learning and lexicon based methods for sentiment classification: A survey*, Proceedings of the 11th Web Information System and Application Conference, Tianjin, pp. 262–265.

[112] Hamad MM & Qader BA, 2014, *Knowledge-driven decision support system based on knowledge warehouse and data mining for market management*, Global Journal of Management And Business Research, **3(10)**, pp. 139–147.

[113] Hampton M, *SVD computational example*, [Lecture Notes], Department of Mathematics and Statistics at the University of Minnesota, Duluth (MN).

[114] Hansen LK & Salamon P, 1990, *Neural network ensembles*, IEEE Transactions on Pattern Analysis and Machine Intelligence, **12**, pp. 993–1001.

[115] Haruechaiyasak C, Kongthon A, Palingoon P & Trakultaweekoon K, 2013, *S-Sense: A sentiment analysis framework for social media sensing*, Proceedings of the Workshop on Natural Language Processing for Social Media, Nagoya, pp. 6–13.

[116] Hassan A, Abbasi A & Zeng D, 2013, *Twitter sentiment analysis: A bootstrap ensemble framework*, Proceedings of the International Conference on Social Computing, Washington, D.C., pp. 357–364.

[117] Hatzivassiloglou V & McKeown KR, 1997, *Predicting the semantic orientation of adjectives*, Proceedings of the 35th Annual Meeting on Association for Computational Linguistics, Madrid, pp. 174–181.

[118] Hatzivassiloglou V & Wiebe JM, 2000, *Effects of adjective orientation and gradability on sentence subjectivity*, Proceedings of the 18th Conference on Computational Linguistics, Saarbrücken, pp. 299–305.

[119] Heerman V (Director), 1930, *Animal Crackers*, [Film], Paramount Pictures.

[120] Hervé A, 2007, *The eigen-decomposition: Eigenvalues and eigenvectors*, pp. 1–10 in Salkind NJ (Ed), *Encyclopedia of Measurements and Statistics*, Sage Publications, Thousand Oaks (CA).

[121] Hestens MR & Stiefel E, 1952, *Methods of conjugate gradients for solving linear systems*, Journal of Research of the National Bureau of Standards, **49(6)**, pp. 409–136.

[122] Hillstrom K, 2005, *The Internet Revolution*, Omnigraphics, Detroit (MI).

[123] Hinckley D, 2017, *New study: Data reveals 67% of consumers are influenced by online reviews*, [Online], [Cited February 2018], Available from `https://moz.com/blog/new-data-reveals-67-of-consumers-are-influenced-by-online-reviews`.

[124] Hochreiter S & Schmidhuber J, 1997, *Long short-term memory*, Neural Computation, **9(8)**, pp. 1735–1780.

[125] Hoffman M, Bach FR & Blei DM, 2010, *Online learning for latent Dirichlet allocation*, Proceedings of the Conference on Advances in Neural Information Processing Systems (NIPS), Vancouver, pp. 856–864.

[126] Hogue D, 2010, *Five essential principles of interaction design*, [Online], [Cited March 2019], Available from `http://www.idux.com/downloads/MAX2010_HOGUE_Interaction_Design_Principles.pdf`.

[127] Holsapple CW, 2008, *DSS architecture and types*, *Handbook on Decision Support Systems 1: Basic Themes*, Springer, Berlin.

[128] Hsueh N-L, 2016, *A framework for opinion mining system with design pattern*, Proceedings of the 2016 International Computer Symposium (ICS), Chiayi, pp. 626–631.

[129] Hu M & Liu B, 2004, *Mining and summarizing customer reviews*, Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle (WA), pp. 168–177.

[130] Hu X & Liu H, 2012, *Text analytics in social media*, pp. 385–414 in Aggarwal CC & Zhai C (Eds), *Text Mining*, Springer Science & Business Media, New York (NY).

[131] Hutto CJ & Gilbert E, 2014, *Vader: A parsimonious rule-based model for sentiment analysis of social media text*, Proceedings of the 8th International AAAI Conference on Weblogs and Social Media, Ann Arbor (MI), pp. 216–225.

[132] Indurkhya N & Damerau FJ (Eds), 2010, *Handbook of Natural Language Processing*, 2nd Edition, Chapman & Hall/CRC, Boca Raton (FL).

[133] Ioffe S & Szegedy C, 2015, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, Proceedings of the 32nd International Conference on Machine Learning, Lille, pp. 448–456.

[134] Isah H, Trundle P & Neagu D, 2014, *Social media analysis for product safety using text mining and sentiment analysis*, Proceedings of the 14th UK Workshop on Computational Intelligence, Bradford, pp. 1–7.

[135] James G, Witten D, Hastie T & Tibshirani R, 2013, *An Introduction to Statistical Learning*, 6th Edition, Springer, New York (NY).

[136] James TL, Villacis Calderon ED & Cook DF, 2017, *Exploring patient perceptions of healthcare service quality through analysis of unstructured feedback*, Expert Systems with Applications, **71**, pp. 479–492.

[137] JAX Editorial Team, 2016, *Python is the most popular programming language today for machine learning*, [Online], [Cited July 2019], Available from `https://jaxenter.com/python-popular-programming-language-today-machine-learning-128942.html`.

[138] JetBrains, 2018, *PyCharm*, [Online], [Cited July 2019], Available from `https://www.jetbrains.com/pycharm/`.

[139] Jiang M-F, Tseng S-S & Liao S-Y, 1999, *Data types generalization for data mining algorithm*, Proceedings of the International Conference on Systems, Man, and Cybernetics, Tokyo, pp. 928–933.

[140] Johnson R & Zhang T, 2015, *Effective use of word order for text categorization with convolutional neural networks*, Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver (CO), pp. 103–112.

[141] Kadkhodaei H & Moghadam AME, 2016, *An entropy based approach to find the best combination of the base classifiers in ensemble classifiers based on stack generalization*, Proceedings of the International Conference on Control, Instrumentation, and Automation, Qazvin, pp. 425–429.

[142] Kamps J, Marx M, Mokken RJ & de Rijke M, 2004, *Using WordNet to measure semantic orientations of adjectives*, Proceedings of the 4th International Conference on Language Resources and Evaluation, Lisbon, pp. 1115–1118.

[143] Kasper W & Vela M, 2012, *Sentiment analysis for hotel reviews*, Speech Technology, **4(11)**, pp. 96–109.

[144] Kazmaier J, 2017, *A machine learning data analysis decision-support system*, BEng Final Year Project, Stellenbosch University, Stellenbosch.

[145] Kazmaier J & van Vuuren JH, 2020, *A generic framework for sentiment analysis: Leveraging opinion-bearing data to inform decision making*, Decision Support Systems, **135**, Manuscript 113304 (no page numbers).

[146] Kazmaier J & van Vuuren JH, 2020, *Sentiment analysis of unstructured customer feedback for a retail bank*, ORiON, **36(1)**, pp. 35–71.

[147] Kendall K & Kendall J, 2013, *Systems Analysis and Design*, 9th Edition, Pearson, Upper Saddle River (NJ).

[148] Kenyon-Dean K, Ahmed E, Fujimoto S, Georges-Filteau J, Glasz C, Kaur B, Lalande A, Bhanderi S, Belfer R & Kanagasabai N, 2018, *Sentiment analysis: It's complicated!*, Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, New Orleans (LA), pp. 1886–1895.

[149] Keras Community, 2016, *Issue with BatchNormalization when used with load_weights*, [Online], [Cited July 2019], Available from `https://github.com/keras-team/keras/issues/3423`.

[150] Keras Community, 2019, *BatchNormalization layer gives inconsistent output*, [Online], [Cited July 2019], Available from `https://github.com/keras-team/keras/issues/12400`.

[151] Keras Team, 2019, *Keras: The Python deep learning library*, [Online], [Cited July 2019], Available from `https://keras.io/`.

[152] Khan FH, Bashir S & Qamar U, 2013, *TOM: Twitter opinion mining framework using hybrid classification scheme*, Decision Support Systems, **57**, pp. 245–257.

[153] Khan FH, Qamar U & Bashir S, 2016, *eSAP: A decision support framework for enhanced sentiment analysis and polarity classification*, Information Sciences, **367**, pp. 862–873.

[154] Khan FH, Qamar U & Bashir S, 2016, *Multi-Objective Model Selection (MOMS)-based semi-supervised framework for sentiment analysis*, Cognitive Computation, **8(4)**, pp. 614–628.

[155] Khan FH, Qamar U & Bashir S, 2016, *SentiMI: Introducing point-wise mutual information with SentiWordNet to improve sentiment polarity detection*, Applied Soft Computing Journal, **39**, pp. 140–153.

[156] Kidd M, 2019, Professor at *the Centre for Statistical Consultiation, Department of Statistics and Actuarial Sciences, Stellenbosch University*, [Personal Communication], Contactable at `mkidd@sun.ac.za`.

[157] Kim S-M & Hovy E, 2006, *Automatic identification of pro and con reasons in online reviews*, Proceedings of the COLING/ACL Main Conference Poster Sessions, Sydney, pp. 483–490.

[158] Kim H & Loh W-Y, 2001, *Classification trees with unbiased multiway splits*, Journal of the American Statistical Association, **96(454)**, pp. 589–604.

[159] Kingma DP & Ba J, 2015, *Adam: A method for stochastic optimization*, Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015), San Diego (CA).

[160] Kiritchenko S, Zhu X & Mohammad SM, 2014, *Sentiment analysis of short informal texts*, Journal of Artificial Intelligence Research, **50**, pp. 723–762.

[161] Knupp J, 2019, *Integrated development environments*, [Online], [Cited March 2014], Available from `https://jeffknupp.com/blog/2014/03/03/what-is-a-web-framework`.

[162] Knutson AL, 1945, *Japanese opinion surveys: The special need and the special difficulties*, Public Opinion Quarterly, **9(3)**, pp. 313–319.

[163] Koch P, Wujek B, Golovidov O & Gardner S, 2017, *Automated hyperparameter tuning for effective machine learning*, Proceedings of the SAS Global Forum 2017 Conference, Orlando (FL), 514:1–514:23.

[164] Ku L-W, Liang Y-T & Chen, H-H, 2006, *Opinion extraction, summarization and tracking in news and blog corpora*, Proceedings of the AAAI Symposium on Computational Approaches to Analysing Weblogs, Boston (MA), pp. 100–107.

[165] Kumar A, McCann R, Naughton J & Patel JM, 2016, *Model selection management systems: The next frontier of advanced analytics*, SIGMOD Record, **44(4)**, pp. 17–22.

[166] Lam W & Lai KY, 2001, *A meta-learning approach for text categorization*, Proceedings of the 24th ACM SIGIR Conference on Research and Development in Information Retrieval, New Orleans (LA), pp. 303–309.

[167] Laurent C, Pereyra G, Brakel P, Zhang Y & Bengio Y, 2016, *Batch normalized recurrent neural networks*, Proceedings of the 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Shanghai, pp. 2657–2661.

[168] le Cun Y, Bengio Y & Hinton G, 2015, *Deep learning*, Nature, **521**, pp. 436–444.

[169] Leal-Taixé L & Niessner M, 2018, *Introduction to deep learning*, [Lecture Notes], Department of Informatics at the Technical University of Munich, Munich.

[170] Ledezma A, Aler R, Sanchis A & Borrajo D, 2010, *GA-stacking: Evolutionary stacked generalization*, Intelligent Data Analysis, **14(1)**, pp. 89–119.

[171] Lee H, Grosse R, Ranganath R & Ng AY, 2009, *Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations*, Proceedings of the 26th Annual International Conference on Machine Learning, Montreal, pp. 609–616.

[172] Li F, Huang M & Zhu X, 2010, *Sentiment analysis with global topics and local dependency*, Proceedings of the AAAI Conference on Artificial Intelligence, Altanta (GA), pp. 1371–1376.

[173] Lin J, 2016, *On the Dirichlet distribution*, Master's Thesis, Queen's University, Ontario.

[174] Lipschutz S & Lipson M, 2009, *Linear Algebra*, 4th Edition, McGraw Hill, New York (NY).

[175] Liu B, 2011, *Opinion mining and sentiment analysis*, pp. 459–526 in Carey MJ & Ceri S (Eds), *Web Data Mining: Exploring Hyperlinks, Contents and Usage Data*, Springer, Berlin.

[176] Liu B, 2012, *Sentiment analysis and opinion mining*, Synthesis Lectures on Human Language Technologies, **1(5)**, pp. 1–168.

[177] Liu B, Hu M & Cheng J, 2005, *Opinion Observer: Analyzing and comparing opinions on the web*, Proceedings of the International World Wide Web Conference, Chiba, pp. 342–351.

[178] Liu B & Zhang L, 2012, *A survey of opinion mining and sentiment analysis*, pp. 415–463 in Aggarwal CC & Zhai CX (Eds), *Mining Text Data*, Springer Science & Business Media, Berlin.

[179] Liu S & Lee I, 2015, *A hybrid sentiment analysis framework for large email data*, Proceedings of the 10th International Conference on Intelligent Systems and Knowledge Engineering, Taipei, pp. 324–330.

[180] Lloret E, Balahur A, Gómez JM, Montoyo A & Palomar M, 2012, *Towards a unified framework for opinion retrieval, mining and summarization*, Journal of Intelligent Information Systems, **39(3)**, pp. 711–747.

[181] Loh W-Y, 2011, *Classification and regression trees*, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, **1(1)**, pp. 14–23.

[182] Luenberger DG & Ye Y, 2008, *Linear and Nonlinear Programming*, 3rd Edition, Springer Science & Business Media, New York (NY).

[183] Maas AL, Daly RE, Pham PT, Huang D, Ng AY & Potts C, 2011, *Learning word vectors for sentiment analysis*, Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Portland (OR), pp. 142–150.

[184] Malhotra MK, Sharma S & Nair SS, 1999, *Decision making using multiple models*, European Journal of Operational Research, **114(1)**, pp. 1–14.

[185] Manning CD, Raghavan P & Schütze H, 2008, *Introduction to Information Retrieval*, Cambridge University Press, New York (NY).

[186] Mäntylä MV, Graziotin D & Kuutila M, 2018, *The evolution of sentiment analysis — A review of research topics, venues, and top cited papers*, Computer Science Review, **27**, pp. 16–32.

[187] Mapbox, *API documentation*, [Online], [Cited July 2019], Available from `https://docs.mapbox.com/api/`.

[188] Mardle S & Pascoe S, 1999, *An overview of genetic algorithms for the solution of optimisation problems*, Computers in Higher Education Economics Review, **13(1)**, pp. 16–20.

[189] McKay MD, 1992, *Latin hypercube sampling as a tool in uncertainty analysis of computer models*, Proceedings of the 24th Conference on Winter Simulation (WSC 1992), New York (NY), pp. 557–564.

[190] Medhat W, Hassan A & Korashy H, 2014, *Sentiment analysis algorithms and applications: A survey*, Ain Shams Engineering Journal, **5(4)**, pp. 1093–1113.

[191] Merriam-Webster, *Opinion*, [Online], [Cited February 2018], Available from `https://www.merriam-webster.com/dictionary/opinion`.

[192] Merriam-Webster, *Sentiment*, [Online], [Cited February 2018], Available from `https://www.merriam-webster.com/dictionary/sentiment`.

[193] Meza JC, 2011, *Newton's method*, Wiley Interdisciplinary Reviews: Computational Statistics, **3(1)**, pp. 75–78.

[194] Mikolov T, Chen K, Corrado G & Dean J, 2013, *Efficient estimation of word representations in vector space*, Proceedings of the International Conference on Learning Representations, Scottsdale (AZ), pp. 1–12.

[195] Mikolov T, Sutskever I, Chen K, Corrado G & Dean J, 2013, *Distributed representations of words and phrases and their compositionality*, Proceedings of the Conference on Neural Information Processing Systems, Lake Tahoe (CA), pp. 1–9.

[196] Miller GA, 1995, *WordNet: A lexical database for English*, Communications of the ACM, **38(11)**, pp. 39–41.

[197] Mitchell M, 1996, *An Introduction to Genetic Algorithms*, MIT Press, Cambridge (MA).

[198] Mitchell TM, 1997, *Machine Learning*, McGraw-Hill Science/Engineering/Math, New York (NY).

[199] Mohammad RA, 2006, *Dilemma between the structured and object-orientated approaches to systems analysis and design*, Journal of Computer Information Systems, **46(3)**, pp. 32–42.

[200] Montgomery DC & Runger GC, 2014, *Applied Statistics and Probability for Engineers*, 6th Edition, John Wiley & Sons, Singapore.

[201] Moraes R, Valiati JF & Gavião Neto WP, 2013, *Document-level sentiment classification: An empirical comparison between SVM and ANN*, Expert Systems with Applications, **40(2)**, pp. 621–633.

[202] Morinaga S, Yamanishi K, Tateishi K & Fukushima T, 2002, *Mining product reputations on the web*, Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Edmonton, pp. 341–349.

[203] Mount J, 2011, *The equivalence of logistic regression and maximum entropy models*, [Online], [Cited November 2018], Available from `http://www.win-vector.com/dfiles/LogisticRegressionMaxEnt.pdf`.

[204] Mozetič I, Grčar M & Smailović J, 2016, *Multilingual Twitter sentiment classification: The role of human annotators*, PLoS ONE, **11(5)**, pp. 1–26.

[205] Mullen T & Collier N, 2004, *Sentiment analysis using support vector machines with diverse information sources*, Proceedings of the 9th Conference on Empirical Methods in Natural Language Processing, Barcelona, pp. 412–418.

[206] Munoz A, 2014, *Machine learning and optimization*, (Unpublished) Technical Report, Courant Institute of Mathematical Sciences, New York (NY).

[207] Murphy KP, 2012, *Machine Learning: A Probabilistic Perspective*, 4th Edition, MIT Press, Cambridge (MA).

[208] Murphy KP, 2006, *Naive Bayes classifiers*, [Lecture Notes], Department of Computer Science at the University of British Columbia, Vancouver.

[209] Na J-C, Sui H, Khoo C, Chan S, Zhou Y, Sui H, Khoo C, Chan S & Zhou Y, 2004, *Effectiveness of simple linguistic processing in automatic sentiment classification of product reviews*, Proceedings of the 8th International Conference of the Society for Knowledge Organization, Würzburg, pp. 49–54.

[210] Naylor TH & Finger JM, 1967, *Verification of computer simulation models*, Management Science, **14(2)**, pp. 92–106.

[211] Németh L, 2019, *Hunspell*, [Online], [Cited May 2019], Available from `https://github.com/hunspell/hunspell`.

[212] Neter J, Kutner MH, Nachtsheim CJ & Wasserman W, 1996, *Applied Linear Statistical Models*, 4th Edition, McGraw Hill/Irwin, New York (NY).

[213] Ng A, 2018, *Machine learning part V: Support vector machines*, [Lecure notes], Department of Computer Science at Stanford University, Stanford (CA).

[214] Ng KW, Tian G-L & Tang M-L, 2011, *Dirichlet and Related Distributions: Theory, Methods and Applications*, John Wiley & Sons, Chichester.

[215] Nguyen DQ, Nguyen DQ, Vu T & Pham SB, 2014, *Sentiment classification on polarity reviews: An empirical study using rating-based features*, Proceedings of the Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis, Baltimore (MD), pp. 128–135.

[216] Nisbet R, Elder J & Miner G, 2009, *Handbook of Statistical Analysis and Data Mining Applications*, Academic Press, Orlando (FL).

[217] Noel S, 1999, *Text mining: The state of the art and the challenges*, Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining 1999 Workshop on Knowledge Disocovery from Advanced Databases, Beijing, pp. 65–70.

[218] Obbayi SR, 2019, *The major types of database management systems*, [Online], [Cited March 2011], Available from `https://www.brighthub.com/internet/web-development/articles/110654.aspx`.

[219] Olah C, 2015, *Understanding LSTM networks*, [Online], [Cited November 2018], Available from `colah.github.io/posts/2015-08-Understanding-LSTMs`.

[220] Omar N, Albared M, Al-Shabi AQ & Al-Moslmi T, 2013, *Ensemble of classification algorithms for subjectivity and sentiment analysis of Arabic customer reviews*, International Journal of Advancements in Computing Technology, **5(12)**, pp. 77–85.

[221] Onan A, 2018, *Particle swarm optimization based stacking method with an application to text classification*, Academic Platform Journal of Engineering and Science, **6(2)**, pp. 134–141.

[222] O'Neil C & Schutt R, 2014, *Doing Data Science: Straight Talk from the Frontline*, 1st Edition, O'Reilly Media Inc., Sebastopol (CA).

[223] Opitz DW & Shavlik JW, 1996, *Actively searching for an effective neural network ensemble*, Connection Science, **8(3–4)**, pp. 337–354.

[224] Oquab M, Bottou L, Laptev I & Sivic J, 2014, *Learning and transferring mid-level image representations using convolutional neural networks*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Columbus (OH), pp. 1717–1724.

[225] Ordenes FV, Theodoulidis B, Burton J, Gruber T & Zaki M, 2014, *Analyzing customer experience feedback using text mining: A linguistics-based approach*, Journal of Service Research, **17(3)**, pp. 278–295.

[226] Ordóñez FJ, Ledezma A & Sanchis A, 2008, *Genetic approach for optimizing ensembles of classifiers*, Proceedings of the International Florida Artificial Intelligence Research Society Conference, Coconut Grove (FL), pp. 89–94.

[227] al-Otaibi S, Alnassar A, Alshahrani A, al-Mubarak A, Albugami S, Almutiri N & Albugami A, 2018, *Customer satisfaction measurement using sentiment analysis*, International Journal of Advanced Computer Science and Applications, **9(2)**, pp. 106–117.

[228] Oxford University Press, 2017, *Definition of data in English*, [Online], [Cited April 2017], Available from `https://en.oxforddictionaries.com/definition/data`.

[229] Oxford University Press, 2018, *Definition of parse in English*, [Online], [Cited March 2018], Available from `https://en.oxforddictionaries.com/definition/parse`.

[230] Pang B & Lee L, 2004, *A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts*, Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics, Barcelona, pp. 271–278.

[231] Pang B & Lee L, 2008, *Opinion mining and sentiment analysis*, Foundations and Trends in Information Retrieval, **2(2)**, pp. 1–135.

[232] Pang B, Lee L & Vaithyanathan S, 2002, *Thumbs up? Sentiment classification using machine learning techniques*, Proceedings of the ACL Conference on Empirical Methods in Natural Language Processing (EMNLP), **10(7)**, pp. 79–86.

[233] Papakonstantinou JM, 2009, *Historical development of the BFGS secant method and its characterization properties*, PhD Thesis, Rice University, Houston (TX).

[234] Papandreou G, Chen L-C, Murphy KP & Yuille AL, 2015, *Weakly- and semi-supervised learning of a deep convolutional network for semantic image segmentation*, Proceedings of the IEEE International Conference on Computer Vision, Las Condes, pp. 1742–1750.

[235] Pascanu R, Mikolov T & Bengio Y, 2013, *On the difficulty of training recurrent neural networks*, Proceedings of the International Conference on Machine Learning (ICML), Atlanta (GA), pp. 1310–1318.

[236] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M & Duchesnay E, 2011, *Scikit-learn: Machine learning in Python*, Journal of Machine Learning Research, **12**, pp. 2825–2830.

[237] Pelletier FJ, 1994, *The principle of semantic compositionality*, Topoi, **13(1)**, pp. 11–24.

[238] Piatetsky G, 2017, *Python overtakes R, becomes the leader in data science, machine learning platforms*, [Online], [Cited July 2019], Available from `http://www.kdnuggets.com/2017/08/python-overtakes-r-leader-analytics-data-science.html`.

[239] Platt JC, 1998, *Sequential minimal optimization: A fast algorithm for training support vector machines*, (Unpublished) Technical Report, Microsoft Research, Washington (WA).

[240] Plotly, 2019, *Dash user guide*, [Online], [Cited July 2019], Available from `https://dash.plot.ly/`.

[241] Polyak BT, 1964, *Some methods of speeding up the convergence of iteration methods*, USSR Computational Mathematics and Mathematical Physics, **4(5)**, pp. 1–17.

[242] Poria S, Cambria E, Winterstein G & Huang GB, 2014, *Sentic patterns: Dependency-based rules for concept-level sentiment analysis*, Knowledge-Based Systems, **69(1)**, pp. 45–63.

[243] Portilla JM, 2017, *Python for data science and machine learning bootcamp*, [Online Course], Udemy.com, URL: `udemy.com/python-for-data-science-and-machine-learning-bootcamp`.

[244] Power DJ, 2000, *Supporting business decision-making*, pp. 1–22 in Power DJ (Ed), *Decision Support Systems Hyperbook*, DSSResources.COM, Cedar Falls (IA).

[245] Power DJ, 2002, *Decision Support Systems: Concepts and Resources for Managers*, Quorom Books, Westport (CT).

[246] Powers D, 2011, *Evaluation: From precision, recall and F-factor to ROC, informedness, markedness & correlation*, Journal of Machine Learning Technologies, **2(1)**, pp. 37–63.

[247] Prabowo R & Thelwall M, 2009, *Sentiment analysis: A combined approach*, Journal of Informetrics, **3(2)**, pp. 143–157.

[248] Prechelt L, 1998, *Early stopping — but when?*, pp. 55–69 in Montavon G, Orr GB & Müller K-R (Eds), *Neural Networks: Tricks of the Trade*, Springer, Berlin.

[249] Pythiest, 2017, *Micro average versus macro average performance in a multiclass classification setting*, [Online], [Cited July 2019], Available from `https://datascience.stackexchange.com/questions/15989/micro-average-vs-macro-average-performance-in-a-multiclass-classification-settin`.

[250] Qiu G, He X, Zhang F, Shi Y, Bu J & Chen C, 2010, *DASA: Dissatisfaction-oriented advertising based on sentiment analysis*, Expert Systems with Applications, **37(9)**, pp. 6182–6191.

[251] Quinlan JR, 1993, *C4. 5: Programs for Machine Learning*, Morgan Kaufmann Publishers, San Mateo (CA).

[252] Radev DR, Teufel S, Saggion H, Lam W, Blitzer J, Qi H, Celebi A, Liu D & Drabek E, 2003, *Evaluation challenges in large-scale document summarization*, Proceedings of the 41st Annual Meeting on Association for Computational Linguistics, Sapporo, pp. 375–382.

[253] Rao Y, Li Q, Mao X & Wenyin L, 2014, *Sentiment topic models for social emotion mining*, Information Sciences, **266**, pp. 90–100.

[254] Rasmus A, Berglund M, Honkala M, Valpola H & Raiko T, 2015, *Semi-supervised learning with ladder networks*, Proceedings of the Conference on Advances in Neural Information Processing Systems (NIPS), Montréal, pp. 3546–3554.

[255] Ravi K & Ravi V, 2015, *A survey on opinion mining and sentiment analysis: Tasks, approaches and applications*, Knowledge-Based Systems, **89**, pp. 14–46.

[256] Řehůřek R, 2019, *Latent Dirichlet allocation*, [Online], [Cited July 2019], Available from `https://radimrehurek.com/gensim/models/ldamodel.html`.

[257] Reinhold J, 2019, *Dropout on convolutional layers is weird*, [Online], [Cited July 2019], Available from `https://towardsdatascience.com/dropout-on-convolutional-layers-is-weird-5c6ab14f19b2`.

[258] Ren G & Hong T, 2017, *Investigating online destination images using a topic-based sentiment analysis approach*, Sustainability, **9(10)**, pp. 1765:1–1765:18.

[259] Robbins H & Monro S, 1951, *A stochastic approximation method*, Annals of Mathematical Statistics, **22(3)**, pp. 400–407.

[260] Rokach L, 2010, *Pattern Classification using Ensemble Methods*, 1st Edition, World Scientific Publishing Company, Singapore.

[261] Rokach L & Maimon O, 2005, *Decision trees*, pp. 165–192 in Rokach L & Maimon OZ (Eds), *Data Mining and Knowledge Discovery Handbook*, Springer, Boston (MA).

[262] Rooney N, Patterson D & Nugent C, 2004, *Reduced ensemble size stacking*, Proceedings of the International Conference on Tools with Artificial Intelligence, Boca Raton (FL), pp. 266–271.

[263] Rowland T & Weisstein EW, 2005, *Tensor*, [Online], [Cited March 2019], Available from `http://mathworld.wolfram.com/Tensor.html`.

[264] Ruder S, 2016, *An overview of gradient descent optimization algorithms*, arXiv Preprint arXiv:1609.04747.

[265] Rumelhart DE, Hinton GE & Williams RJ, 1986, *Learning representations by back-propagating errors*, Nature, **323**, pp. 533–536.

[266] Sagi O & Rokach L, 2018, *Ensemble learning: A survey*, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, **8(4)**, pp. 1–18.

[267] Salinca A, 2015, *Business reviews classification using sentiment analysis*, Proceedings of the International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), Timisoara, pp. 247–250.

[268] Samir M, 2016, *Machine learning theory — Part 2: Generalization bounds*, [Online], [Cited November 2018], Available from `https://mostafa-samir.github.io/ml-theory-pt2`.

[269] Sargent RG, 2010, *Verification and validation of simulation models*, Proceedings of the Winter Simulation Conference, Baltimore (MD), pp. 166–183.

[270] Sastry K, Goldberg D & Kendall G, 2005, *Genetic algorithms*, pp. 97–125Burke EK & Kendall G (Eds), *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, Springer US, Boston (MA).

[271] Satzinger JW, Jackson RB & Burd SD, 2012, *Systems Analysis and Design in a Changing World*, 6th Edition, Cengage Learning, Boston (MA).

[272] Schafer JL & Graham JW, 2002, *Missing data: Our view of the state of the art*, Psychological Methods, **7(2)**, pp. 147–77.

[273] Schapire RE, 1990, *The strength of weak learnability*, Machine Language, **5(2)**, pp. 197–227.

[274] Schmidt M, Fung G & Rosales R, 2007, *Fast optimization methods for L1 regularization: A comparative study and two new approaches*, Proceedings of the 18th European Conference on Machine Learning, Warsaw, pp. 286–297.

[275] Schouten K, Frasincar F, Dekker R & Riezebos M, 2019, *Heracles: A framework for developing and evaluating text mining algorithms*, Expert Systems with Applications, **127**, pp. 68–84.

[276] Schreyer A, 2006, *GA optimization for Excel version 1.2*, [Online], [Cited May 2020], Available from `https://alexschreyer.net/wp-content/uploads/2006/12/ga_optimization_for_excel_1_2.pdf`.

[277] Seewald A, 2002, *How to make stacking better and faster while also taking care of an unknown weakness*, Proceedings of the 19th International Conference on Machine Learning, pp. 554–561.

[278] Sesmero MP, Ledezma AI & Sanchis A, 2015, *Generating ensembles of heterogeneous classifiers using stacked generalization*, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, **5(1)**, pp. 21–34.

[279] Shanno DF, 1970, *Conditioning of quasi-Newton methods for function minimization*, Mathematics of Computation, **24(111)**, pp. 647–656.

[280] Sharma A & Dey S, 2012, *A comparative study of feature selection and machine learning techniques for sentiment analysis*, Proceedings of the 2012 ACM Research in Applied Computation Symposium (RACS), San Antonio (TX), pp. 1–7.

[281] Shim J, Warkentin M, Courtney JF, Power DJ, Sharda R & Carlsson C, 2002, *Past, present, and future of decision support technology*, Decision Support Systems, **33**, pp. 111–126.

[282] Shimodaira H, 2015, *Text classification using naive Bayes*, [Lecture Notes], The University of Edinburgh School of Informatics, Edinburgh.

[283] Shiotsu Y, 2019, *Web development 101: Top web development languages in 2014*, [Online], [Cited March 2014], Available from `https://www.upwork.com/blog/2014/03/web-development-101-top-web-development-languages-2014/`.

[284] Shook E, Leetaru K, Cao G, Padmanabhan A & Wang S, 2012, *Happy or not: Generating topic-based emotional heatmaps for culturomics using CyberGIS*, Proceedings of the 8th International Conference on E-Science, Chicago (IL), pp. 1–6.

[285] Shunmugapriya P & Kanmani S, 2013, *Optimization of stacking ensemble configurations through artificial bee colony algorithm*, Swarm and Evolutionary Computation, **12**, pp. 24–32.

[286] Sievert C & Shirley K, 2015, *LDAvis: Details*, [Online], [Cited July 2019], Available from `https://cran.r-project.org/web/packages/LDAvis/vignettes/details.pdf`.

[287] Sievert C & Shirley K, 2014, *LDAvis: A method for visualizing and interpreting topics*, Proceedings of the Workshop on Interactive Language Learning, Visualization, and Interfaces, Baltimore (MD), pp. 63–70.

[288] Simon H, 2017, *Decision support systems*, pp. 31–69 in Filip FG, Zamfirescu C-B & Ciurea C (Eds), *Computer-Supported Collaborative Decision-Making*, Springer International Publishing, New York (NY).

[289] Sipser M, 2006, *Introduction to the Theory of Computation*, 2nd Edition, Thomson Course Technology, Boston (MA).

[290] Skiena SS, 2017, *The algorithm design manual*, [Online], [Cited February 2018], Available from `http://www2.toki.or.id/book/algdesignmanual/index.htm`.

[291] Software Testing Help, 2019, *Top 30 most popular database management software: Complete list*, [Online], [Cited March 2018], Available from `https://www.softwaretestinghelp.com/database-management-software/`.

[292] Sperling G, 2017, *South African report delivers insight into influence of mobile on consumer behaviour*, [Online], [Cited February 2018], Available from `http://www.bizni safrica.com/south-african-report-delivers-insight-influence-mobile-consumer-behaviour/`.

[293] Spolsky J, 2011, *User Interface Design for Programmers*, Apress, New York (NY).

[294] Srivastava N, Hinton G, Krizhevsky A, Sutskever I & Salakhutdinov R, 2014, *Dropout: A simple way to prevent neural networks from overfitting*, Journal of Machine Learning Research, **15(1)**, pp. 1929–1958.

[295] StackOverflow, 2016, *What are the major differences and benefits of Porter and Lancaster stemming algorithms?*, [Online], [Cited July 2019], Available from `https://stackoverflow.com/questions/10554052/what-are-the-major-differences-and-benefits-of-porter-and-lancaster-stemming-alg`.

[296] Stair RM, Reynolds GW, Hulbert J, Calhoun JW & Shipp L, 2012, *Principles of Information Systems*, 10th Edition, Cengage Learning, Boston (MA).

[297] Stanford University, 2018, *Convolutional neural networks for visual recognition: Image classification*, [Lecture Notes], Department of Computer Science at Stanford University, Stanford (CA).

[298] Stats SA, *Census 2011*, [Online], [Cited July 2019], Available from `https://www.statssa.gov.za/publications/P03014/P030142011.pdf`.

[299] Stewart J, 2012, *Calculus*, 7th Edition, Brooks/Cole, Cengange Learning, Pacific Grove (CA).

[300] Steynberg R, 2016, *Framework for identifying the most likely to be successful underprivileged tertiary bursary applicants*, Master's Thesis, Stellenbosch University, Stellenbosch.

[301] Stotts D, 2000, *Intramodule cohesion*, [Lecture Notes], Department of Computer Science at the University of North Carolina, Chapel Hill (NC).

[302] Stover C & Weisstein EW, 2018, *Closed-form solution*, [Online], [Cited November 2018], Available from `http://mathworld.wolfram.com/Closed-FormSolution.html`.

[303] Strang G, 2006, *Linear Algebra and its Applications*, 4th Edition, Thomson Brooks/Cole, Belmont (CA).

[304] Strang G, 2009, *Introduction to Linear Algebra*, 4th Edition, Wellesley Cambridge Press, Wellesley (MA).

[305] Stützle T, 1999, *Local search algorithms for combinatorial problems — Analysis, improvements, and new applications*, PhD Thesis, Technical University of Darmstadt, Darmstadt.

[306] Sun S, Luo C & Chen J, 2017, *A review of natural language processing techniques for opinion mining systems*, Information Fusion, **36**, pp. 10–25.

[307] Surowiecki J, 2004, *The Wisdom of Crowds: Why the Many Are Smarter than the Few and How Collective Wisdom Shapes Business, Economies, Societies, and Nations*, Doubleday & Co, New York (NY).

[308] Tang D, Qin B & Liu T, 2015, *Document modeling with gated recurrent neural network for sentiment classification*, Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, Lisbon, pp. 1422–1432.

[309] TensorFlow, 2018, *Vector representations of words*, [Online], [Cited August 2018], Available from `https://www.tensorflow.org/tutorials/representation/word2vec`.

[310] TensorFlow, 2019, *An end-to-end open source machine learning platform*, [Online], [Cited July 2019], Available from `https://www.tensorflow.org`.

[311] TensorFlow, 2019, *TensorBoard: Visualizing learning*, [Online], [Cited July 2019], Available from `https://www.tensorflow.org/guide/summaries%5C_and%5C_tensorboard`.

[312] The Qt Company Ltd., 2019, *Qt designer manual*, [Online], [Cited July 2019], Available from `https://doc.qt.io/qt-5/qtdesigner-manual.html`.

[313] The Qt Company Ltd., 2019, *Qt documentation*, [Online], [Cited July 2019], Available from `https://doc.qt.io/qt-5/index.html`.

[314] Ting KM & Witten IH, 1999, *Issues in stacked generalization*, Journal of Artificial Intelligence Research, **10**, pp. 271–289.

[315] Tompson J, Goroshin R, Jain A, le Cun Y & Bregler C, 2015, *Efficient object localization using convolutional networks*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston (MA), pp. 648–656.

[316] Tripathy A, Agrawal A & Rath SK, 2016, *Classification of sentiment reviews using n-gram machine learning approach*, Expert Systems with Applications, **57**, pp. 117–126.

[317] Tsoumakas G, Partalas I & Vlahavas I, 2008, *A taxonomy and short review of ensemble selection*, Proceedings of the European Conference on Artificial Intelligence, Patras, pp. 41–46.

[318] Tsutsumi K, Shimada K & Endo T, 2007, *Movie review classification based on a multiple classifier*, Proceedings of the Pacific Asia Conference on Language, Information and Computation, Taipei, pp. 481–488.

[319] Tsytsarau M & Palpanas T, 2012, *Survey on mining subjective data on the web*, Data Mining and Knowledge Discovery, **24(3)**, pp. 478–514.

[320] Tuck S, Alkuraiji A, Smith MH, Liu S, Pan J & Jayawickrama U, 2014, *Where can knowledge-based decision support systems go in contemporary business management — A new architecture for the future*, Journal of Economics, Business and Management, **3(5)**, pp. 498–504.

[321] Turney PD, 2002, *Thumbs up or thumbs down? Semantic orientation applied to unsupervised classification of reviews*, Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL), Philadelphia (PA), pp. 417–424.

[322] Turney PD & Littman ML, 2003, *Measuring praise and criticism*, ACM Transactions on Information Systems, **21(4)**, pp. 315–346.

[323] van den Bos A, 2007, *Parameter Estimation for Scientists and Engineers*, John Wiley & Sons, Hoboken (NJ).

[324] Van Rijsbergen CJ, Robertson SE & Porter MF, 1980, *New Models in Probabilistic Information Retrieval*, British Library Research and Development Department, London.

[325] Vargas JA, 2012, *Spring awakening: How an Egyptian revolution began on Facebook*, [Online], [Cited February 2018], Available from `http://www.nytimes.com/2012/02/19/books/review/how-an-egyptian-revolution-began-on-facebook.html`.

[326] Vincent P, Larochelle H, Bengio Y & Manzagol P-A, 2008, *Extracting and composing robust features with denoising autoencoders*, Proceedings of the 25th International Conference on Machine Learning (ICML), New York (NY), pp. 1096–1103.

[327] Vincent P, Larochelle H, Lajoie I, Bengio Y & Manzagol P-A, 2010, *Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion*, Journal of Machine Learning Research, **11**, pp. 3371–3408.

[328] Vural AG, Cambazoglu BB, Senkul P & Tokgoz ZO, 2013, *A Framework for sentiment analysis in Turkish: Application to polarity detection of movie reviews in Turkish*, Computer and Information Sciences, **3**, pp. 437–445.

[329] Walker MA, Anand P, Abbott R, Tree JEF, Martell C & King J, 2012, *That is your evidence?: Classifying stance in online political debate*, Decision Support Systems, **53(4)**, pp. 719–729.

[330] Wang G, Sun J, Ma J, Xu K & Gu J, 2014, *Sentiment classification: The contribution of ensemble learning*, Decision Support Systems, **57(1)**, pp. 77–93.

[331] Wang S & Manning C, 2012, *Baselines and bigrams: Simple, good sentiment and topic classification*, Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics, Jeju Island, pp. 90–94.

[332] We3Schools, 2019, *What is JSON?*, [Online], [Cited July 2019], Available from `https://www.w3schools.com/whatis/whatis%5C_json.asp`.

[333] Weiss SM, Indurkhya N & Zhang T, 2010, *From textual information to numerical vectors*, pp. 13–38 in Indurkhya N, Weiss SM & Zhang T (Eds), *Fundamentals of Predictive Text Mining*, Springer, London.

[334] Weiss SM, Indurkhya N, Zhang T & Damerau FJ, 2005, *Overview of text mining*, pp. 1–13 in Zhang T, Damerau F, Indurkhya N & Weiss SM (Eds), *Text Mining: Predictive Methods for Analyzing Unstructured Information*, Springer, London.

[335] Weisstein EW, 2010, *Maximum likelihood*, [Online], [Cited August 2018], Available from `http://mathworld.wolfram.com/MaximumLikelihood.html`.

[336] Weisstein EW, 2018, *Method of steepest descent*, [Online], [Cited November 2018], Available from `http://mathworld.wolfram.com/MethodofSteepestDescent.html`.

[337] Weisstein EW, 2018, *Simplex*, [Online], [Cited November 2018], Available from `http://mathworld.wolfram.com/Simplex.html`.

[338] Whalen S & Pandey G, 2013, *A comparative analysis of ensemble classifiers: Case studies in genomics*, Proceedings of the IEEE International Conference on Data Mining, Dallas (TX), pp. 807–816.

[339] Wiebe JM, 1990, *Recognizing subjective sentences: A computational investigation of narrative text*, PhD Thesis, State University of New York at Buffalo, Buffalo (NY).

[340] Wilson T, Wiebe J & Hoffmann P, 2005, *Recognizing contextual polarity in phrase-level sentiment analysis*, Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing, Vancouver, pp. 347–354.

[341] Wilson T, Wiebe J & Hwa R, 2006, *Recognizing strong and weak opinon clauses*, Computational Intelligence, **22(2)**, pp. 73–99.

[342] Winston WL & Goldberg JB, 2004, *Operations Research: Applications and Algorithms*, 4th Edition, Brooks/Cole, Belmot (CA).

[343] Wolpert DH, 1992, *Stacked generalization*, Neural Networks, **5(2)**, pp. 241–259.

[344] Wu M, 2010, *A scientist's view of social CRM*, [Online], [Cited February 2018], Available from `https://community.lithium.com/t5/Science-of-Social-Blog/A-Scientists-View-of-Social-CRM/ba-p/7687`.

[345] Wu SJ, Chiang RD & Chang HC, 2018, *Applying sentiment analysis in social web for smart decision support marketing*, Journal of Ambient Intelligence and Humanized Computing, **doi: 10.1007/s12652-018-0683-9**, pp. 1–10.

[346] Xia R, Zong C & Li S, 2011, *Ensemble of feature sets and classification algorithms for sentiment classification*, Information Sciences, **181(6)**, pp. 1138–1152.

[347] Xu J, Chen D, Qiu X & Huang X, 2016, *Cached long short-term memory neural networks for document-level sentiment classification*, Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, Austin (TX), pp. 1660–1669.

[348] Xue JH & Titterington DM, 2008, *Comment on "On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes"*, Neural Processing Letters, **28(3)**, pp. 169–187.

[349] Yaakub MR, Li Y & Zhang J, 2013, *Integration of sentiment analysis into customer relational model: The importance of feature ontology and synonym*, Procedia Technology, **11**, pp. 495–501.

[350] Yadollahi A, Shahraki AG & Zaiane OR, 2017, *Current state of text sentiment analysis from opinion to emotion mining*, ACM Computing Surveys, **50(2)**, pp. 1–33.

[351] Yang Z, Yang D, Dyer C, He X, Smola A & Hovy E, 2016, *Hierarchical attention networks for document classification*, Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT), San Diego (CA), pp. 1480–1489.

[352] Yassine M & Hajj H, 2010, *A framework for emotion mining from text in online social networks*, Proceedings of the IEEE International Conference on Data Mining, Sydney, pp. 1136–1142.

[353] Yi J, Nasukawa T, Bunescu R & Niblack W, 2004, *Sentiment analyzer: Extracting sentiments about a given topic using natural language processing techniques*, Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM'03), Melbourne (FL), pp. 427–434.

[354] Yussupova N, Boyko M & Bogdanova D, 2015, *A decision support approach based on sentiment analysis combined with data mining for customer satisfaction research*, International Journal on Advances in Intelligent Systems, **1(1)**, pp. 145–158.

[355] Zaied ANH, Aal SIA & Hassan MM, 2013, *Rule-based expert systems for selecting information systems development methodologies*, International Journal of Intelligent Systems and Applications, **5(9)**, pp. 19.

[356] Zeil SJ, 2019, *Integrated development environments*, [Online], [Cited March 2017], Available from `https://www.cs.odu.edu/~zeil/cs350/f17/Public/IDEs/index.html`.

[357] Zhai S & Zhang ZM, 2016, *Semisupervised autoencoder for sentiment analysis*, Proceedings of the AAAI Conference on Artificial Intelligence (AAAI-16), Phoenix (AZ), pp. 1394–1400.

[358] Zhang L, Wang S & Liu B, 2018, *Deep learning for sentiment analysis: A survey*, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, **8(4)**, pp. e1253:1–e1253:34.

[359] Zhao H, Gallo O, Frosio I & Kautz J, 2017, *Loss functions for image restoration with neural networks*, IEEE Transactions on Computational Imaging, **3(1)**, pp. 47–57.

[360] Zhou YT & Chellappa R, 1988, *Computation of optical flow using a neural network*, Proceedings of the 1988 International Conference on Neural Networks, San Diego (CA), pp. 71–78.

[361] Zhou Z-H, 2015, *Ensemble learning*, pp. 411–416 in Li SZ & Jain AK (Eds), *Encyclopedia of Biometrics*, Springer US, Boston (MA).

[362] Zhou Z-H, 2012, *Ensemble Methods: Foundations and Algorithms*, 1st Edition, Chapman & Hall/CRC, Baca Raton (FL).

[363] Zhu X, 2010, *Advanced natural language processing: Support vector machines*, [Lecture Notes], Department of Computer Science at the University of Wisconsin-Madison, Madicon (WI).

[364] Zill DG & Wright WS, 2014, *Advanced Engineering Mathematics*, 5th Edition, Jones & Bartlett Learning, Burlington (MA).

APPENDIX A

Detailed modelling results for the case study

The detailed results of each of the ten experimental runs conducted during the model development phase of the case study analysis of Chapter 9 are given in Tables A.1–A.10. More specifically, for each run, the algorithm, its input feature set and the hyperparameter values selected by means of the grid search are given, along with the performance achieved in respect of the accuracy metric during the grid search (the *cross-validated score*), as well as the performance achieved in respect of the AUC and accuracy[1] metric during testing. In order to save space, the *best parameters* for the deep learning algorithms are not shown in full. More specifically, only the parameters for which values were selected by means of the grid search algorithm are shown. The remaining hyperparameter values are as specified in Table 9.2.

Algorithm	Document representation	*n*-gram range	Best parameters	Cross-validated score	Accuracy	AUC
NB	Frequency	(1, 1)	alpha: 0.6	0.804	0.77	0.885
SVM	Frequency	(1, 1)	C: 10.0, gamma: 0.1, kernel: sigmoid	0.825	0.79	0.747
LogReg	Frequency	(1, 1)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.832	0.794	0.886
ANN	Frequency	(1, 1)	max_epochs: 12	0.823	0.806	0.878
NB	Frequency	(2, 2)	alpha: 0.6	0.750	0.750	0.736
SVM	Frequency	(2, 2)	C: 1.0, gamma: 1.0, kernel: rbf	0.751	0.754	0.558
LogReg	Frequency	(2, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.747	0.75	0.74
ANN	Frequency	(2, 2)	max_epochs: 10	0.735	0.752	0.726
NB	Frequency	(1, 2)	alpha: 0.4	0.795	0.758	0.887
SVM	Frequency	(1, 2)	C: 10.0, gamma: 0.1, kernel: sigmoid	0.829	0.796	0.758
LogReg	Frequency	(1, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.833	0.79	0.886
ANN	Frequency	(1, 2)	max_epochs: 10	0.823	0.800	0.873
NB	Frequency	(1, 1)	alpha: 0.2	0.805	0.798	0.876

[1] As explained in §7.2.2, the F1 score, precision and recall are equal to the accuracy score due to micro-averaging.

SVM	Frequency	(1, 1)	C: 10.0, gamma: 0.1, kernel: rbf	0.827	0.808	0.785
LogReg	Frequency	(1, 1)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.834	0.800	0.888
ANN	Frequency	(1, 1)	max_epochs: 10	0.824	0.800	0.882
NB	Frequency	(2, 2)	alpha: 1.0	0.747	0.746	0.704
SVM	Frequency	(2, 2)	C: 1.0, gamma: 1.0, kernel: rbf	0.750	0.7569	0.561
LogReg	Frequency	(2, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.745	0.754	0.745
ANN	Frequency	(2, 2)	max_epochs: 10	0.739	0.750	0.702
NB	Frequency	(1, 2)	alpha: 0.0001	0.797	0.800	0.873
SVM	Frequency	(1, 2)	C: 10.0, gamma: 0.1, kernel: rbf	0.817	0.810	0.781
LogReg	Frequency	(1, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.830	0.798	0.889
ANN	Frequency	(1, 2)	max_epochs: 12	0.823	0.804	0.881
NB	TF-IDF	(1, 1)		0.551	0.48	0.779
SVM	TF-IDF	(1, 1)	C: 10.0, gamma: 0.1, kernel: rbf	0.842	0.828	0.764
LogReg	TF-IDF	(1, 1)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.834	0.804	0.886
ANN	TF-IDF	(1, 1)	max_epochs: 10	0.827	0.796	0.875
NB	TF-IDF	(2, 2)		0.443	0.436	0.690
SVM	TF-IDF	(2, 2)	C: 1.0, gamma: 1.0, kernel: rbf	0.750	0.752	0.557
LogReg	TF-IDF	(2, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.745	0.752	0.747
ANN	TF-IDF	(2, 2)	max_epochs: 12	0.734	0.744	0.719
NB	TF-IDF	(1, 2)		0.561	0.496	0.782
SVM	TF-IDF	(1, 2)	C: 10.0, gamma: 0.1, kernel: rbf	0.834	0.824	0.762
LogReg	TF-IDF	(1, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.827	0.804	0.886
ANN	TF-IDF	(1, 2)	max_epochs	0.816	0.804	0.883
CNN	Embeddings		max_epochs: 12	0.835	0.824	0.889
LSTM	Embeddings		max_epochs: 4	0.835	0.804	0.884
Sentiwordnet					0.496	0.616
Pattern					0.377	0.604
Hu and Liu					0.391	0.597
Vader					0.41	0.587

TABLE A.1: *Detailed case study results for the first experimental run.*

Algorithm	Document representation	*n*-gram range	Best parameters	Cross-validated score	Accuracy	AUC
NB	Presence	(1, 1)	alpha: 0.8	0.81	0.824	0.928
SVM	Presence	(1, 1)	C: 10.0, gamma: 0.1, kernel: rbf	0.819	0.842	0.832

LogReg	Presence	(1, 1)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.82	0.854	0.928
ANN	Presence	(1, 1)	max_epochs: 10	0.801	0.852	0.918
NB	Presence	(2, 2)	alpha: 0.6	0.746	0.756	0.739
SVM	Presence	(2, 2)	C: 1.0, gamma: 1.0, kernel: rbf	0.747	0.756	0.574
LogReg	Presence	(2, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.748	0.762	0.753
ANN	Presence	(2, 2)	max_epochs: 10	0.731	0.754	0.747
NB	Presence	(1, 2)	alpha: 1.0	0.791	0.808	0.927
SVM	Presence	(1, 2)	C: 10.0, gamma: 0.1, kernel: sigmoid	0.814	0.846	0.832
LogReg	Presence	(1, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: newton-cg	0.817	0.848	0.931
ANN	Presence	(1, 2)	max_epochs: 10	0.808	0.834	0.924
NB	Frequency	(1, 1)	alpha: 0.0001	0.795	0.826	0.89
SVM	Frequency	(1, 1)	C: 10.0, gamma: 0.1, kernel: rbf	0.817	0.828	0.824
LogReg	Frequency	(1, 1)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.821	0.84	0.925
ANN	Frequency	(1, 1)	max_epochs: 10	0.808	0.838	0.925
NB	Frequency	(2, 2)	alpha: 1.0	0.741	0.754	0.669
SVM	Frequency	(2, 2)	C: 1.0, gamma: 1.0, kernel: rbf	0.747	0.754	0.575
LogReg	Frequency	(2, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.744	0.756	0.749
ANN	Frequency	(2, 2)	max_epochs: 12	0.728	0.746	0.747
NB	Frequency	(1, 2)	alpha: 0.0001	0.79	0.828	0.897
SVM	Frequency	(1, 2)	C: 10.0, gamma: 0.1, kernel: rbf	0.814	0.838	0.837
LogReg	Frequency	(1, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.817	0.846	0.93
ANN	Frequency	(1, 2)	max_epochs: 10	0.815	0.826	0.916
NB	TF-IDF	(1, 1)		0.542	0.47	0.774
SVM	TF-IDF	(1, 1)	C: 10.0, gamma: 0.1, kernel: rbf	0.836	0.852	0.823
LogReg	TF-IDF	(1, 1)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.822	0.854	0.919
ANN	TF-IDF	(1, 1)	max_epochs: 12	0.814	0.834	0.908
NB	TF-IDF	(2, 2)		0.443	0.412	0.669
SVM	TF-IDF	(2, 2)	C: 1.0, gamma: 1.0, kernel: rbf	0.745	0.754	0.575
LogReg	TF-IDF	(2, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.746	0.756	0.75
ANN	TF-IDF	(2, 2)	max_epochs: 10	0.724	0.756	0.747
NB	TF-IDF	(1, 2)		0.561	0.472	0.785
SVM	TF-IDF	(1, 2)	C: 10.0, gamma: 0.1, kernel: rbf	0.834	0.856	0.828

LogReg	TF-IDF	(1, 2)	C: 10.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.822	0.854	0.924
ANN	TF-IDF	(1, 2)	max_epochs: 10	0.815	0.854	0.919
CNN	Embeddings		max_epochs: 10	0.834	0.858	0.927
LSTM	Embeddings		max_epochs: 4	0.825	0.856	0.927
Sentiwordnet					0.463	0.581
Pattern					0.369	0.597
Hu and Liu					0.377	0.588
Vader					0.408	0.589

TABLE A.2: *Detailed case study results for the second experimental run.*

Algorithm	Document representation	*n*-gram range	Best parameters	Cross-validated score	Accuracy	AUC
NB	Presence	(1, 1)	alpha: 1.0	0.803	0.766	0.848
SVM	Presence	(1, 1)	C: 10.0, gamma: 0.1, kernel: rbf	0.829	0.82	0.784
LogReg	Presence	(1, 1)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.834	0.822	0.861
ANN	Presence	(1, 1)	max_epochs: 12	0.815	0.83	0.86
NB	Presence	(2, 2)	alpha: 0.6	0.749	0.762	0.749
SVM	Presence	(2, 2)	C: 1.0, kernel: linear	0.748	0.756	0.562
LogReg	Presence	(2, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.748	0.76	0.755
ANN	Presence	(2, 2)	max_epochs: 12	0.741	0.756	0.741
NB	Presence	(1, 2)	alpha: 0.6	0.788	0.756	0.849
SVM	Presence	(1, 2)	C: 10.0, gamma: 0.1, kernel: sigmoid	0.831	0.818	0.763
LogReg	Presence	(1, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.83	0.816	0.859
ANN	Presence	(1, 2)	max_epochs: 12	0.802	0.81	0.857
NB	Frequency	(1, 1)	alpha: 0.2	0.811	0.796	0.845
SVM	Frequency	(1, 1)	C: 1.0, kernel: linear	0.828	0.812	0.762
LogReg	Frequency	(1, 1)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.838	0.818	0.862
ANN	Frequency	(1, 1)	max_epochs: 10	0.814	0.794	0.858
NB	Frequency	(2, 2)	alpha: 0.6	0.747	0.758	0.713
SVM	Frequency	(2, 2)	C: 1.0, kernel: linear	0.748	0.752	0.559
LogReg	Frequency	(2, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.748	0.758	0.753
ANN	Frequency	(2, 2)	max_epochs: 10	0.734	0.76	0.738
NB	Frequency	(1, 2)	alpha: 0.2	0.81	0.792	0.843
SVM	Frequency	(1, 2)	C: 10.0, gamma: 0.1, kernel: rbf	0.826	0.822	0.791
LogReg	Frequency	(1, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.828	0.818	0.858
ANN	Frequency	(1, 2)	max_epochs: 10	0.816	0.808	0.853
NB	TF-IDF	(1, 1)		0.58	0.492	0.778

SVM	TF-IDF	(1, 1)	C: 10.0, gamma: 0.1, kernel: rbf	0.84	0.826	0.763
LogReg	TF-IDF	(1, 1)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.83	0.81	0.857
ANN	TF-IDF	(1, 1)	max_epochs: 10	0.818	0.82	0.856
NB	TF-IDF	(2, 2)		0.436	0.44	0.686
SVM	TF-IDF	(2, 2)	C: 1.0, kernel: linear	0.749	0.756	0.567
LogReg	TF-IDF	(2, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.749	0.76	0.749
ANN	TF-IDF	(2, 2)	max_epochs: 10	0.742	0.756	0.731
NB	TF-IDF	(1, 2)		0.586	0.526	0.784
SVM	TF-IDF	(1, 2)	C: 1.0, gamma: 1.0, kernel: rbf	0.84	0.822	0.743
LogReg	TF-IDF	(1, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.825	0.806	0.86
ANN	TF-IDF	(1, 2)	max_epochs: 10	0.817	0.808	0.859
CNN	Embeddings		max_epochs: 10	0.841	0.816	0.866
LSTM	Embeddings		max_epochs: 10	0.832	0.818	0.871
Sentiwordnet					0.465	0.588
Pattern					0.406	0.61
Hu and Liu					0.396	0.602
Vader					0.428	0.592

TABLE A.3: *Detailed case study results for the third experimental run.*

Algorithm	Document representation	*n*-gram range	Best parameters	Cross-validated score	Accuracy	AUC
NB	Presence	(1, 1)	alpha: 0.2	0.798	0.8	0.903
SVM	Presence	(1, 1)	C: 10.0, gamma: 0.1, kernel: sigmoid	0.82	0.838	0.777
LogReg	Presence	(1, 1)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.823	0.838	0.907
ANN	Presence	(1, 1)	max_epochs: 12	0.799	0.828	0.893
NB	Presence	(2, 2)	alpha: 0.4	0.748	0.756	0.744
SVM	Presence	(2, 2)	C: 1.0, gamma: 1.0, kernel: rbf	0.747	0.756	0.564
LogReg	Presence	(2, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.744	0.758	0.738
ANN	Presence	(2, 2)	max_epochs: 10	0.732	0.746	0.724
NB	Presence	(1, 2)	alpha: 1.0	0.788	0.784	0.903
SVM	Presence	(1, 2)	C: 10.0, gamma: 0.1, kernel: rbf	0.814	0.848	0.808
LogReg	Presence	(1, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.813	0.838	0.908
ANN	Presence	(1, 2)	max_epochs: 12	0.797	0.842	0.901
NB	Frequency	(1, 1)	alpha: 0.2	0.804	0.816	0.888
SVM	Frequency	(1, 1)	C: 1.0, gamma: 0.1, kernel: rbf	0.813	0.826	0.708

LogReg	Frequency	(1, 1)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.813	0.836	0.907
ANN	Frequency	(1, 1)	max_epochs: 10	0.809	0.848	0.909
NB	Frequency	(2, 2)	alpha: 0.6	0.742	0.752	0.713
SVM	Frequency	(2, 2)	C: 1.0, gamma: 1.0, kernel: rbf	0.746	0.76	0.574
LogReg	Frequency	(2, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.742	0.758	0.741
ANN	Frequency	(2, 2)	max_epochs: 12	0.736	0.744	0.722
NB	Frequency	(1, 2)	alpha: 0.0001	0.801	0.824	0.879
SVM	Frequency	(1, 2)	C: 10.0, gamma: 0.1, kernel: rbf	0.815	0.844	0.803
LogReg	Frequency	(1, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.811	0.842	0.907
ANN	Frequency	(1, 2)	max_epochs: 12	0.813	0.846	0.906
NB	TF-IDF	(1, 1)		0.568	0.5	0.771
SVM	TF-IDF	(1, 1)	C: 1.0, gamma: 1.0, kernel: rbf	0.84	0.832	0.729
LogReg	TF-IDF	(1, 1)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.82	0.828	0.892
ANN	TF-IDF	(1, 1)	max_epochs: 12	0.814	0.846	0.893
NB	TF-IDF	(2, 2)		0.444	0.42	0.677
SVM	TF-IDF	(2, 2)	C: 1.0, degree: 2, gamma: 1.0, kernel: poly	0.746	0.754	0.57
LogReg	TF-IDF	(2, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.744	0.76	0.745
ANN	TF-IDF	(2, 2)	max_epochs: 12	0.734	0.754	0.735
NB	TF-IDF	(1, 2)		0.579	0.524	0.781
SVM	TF-IDF	(1, 2)	C: 10.0, gamma: 0.1, kernel: rbf	0.84	0.852	0.783
LogReg	TF-IDF	(1, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.82	0.836	0.893
ANN	TF-IDF	(1, 2)	max_epochs: 10	0.813	0.852	0.897
CNN	Embeddings		max_epochs: 10	0.818	0.862	0.914
LSTM	Embeddings	max_epo 4	0.819	0.856	0.913	
Sentiwordnet					0.465	0.587
Pattern					0.416	0.608
Hu and Liu					0.4	0.597
Vader					0.41	0.573

TABLE A.4: *Detailed case study results for the fourth experimental run.*

Algorithm	Document representation	*n*-gram range	Best parameters	Cross-validated score	Accuracy	AUC
NB	Presence	(1, 1)	alpha: 0.6	0.808	0.826	0.898
SVM	Presence	(1, 1)	C: 10.0, gamma: 0.1, kernel: sigmoid	0.823	0.842	0.792

LogReg	Presence	(1, 1)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.831	0.832	0.904
ANN	Presence	(1, 1)	max_epochs: 10	0.812	0.834	0.879
NB	Presence	(2, 2)	alpha: 0.0001	0.744	0.764	0.778
SVM	Presence	(2, 2)	C: 1.0, gamma: 1.0, kernel: rbf	0.746	0.758	0.565
LogReg	Presence	(2, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.743	0.758	0.766
ANN	Presence	(2, 2)	max_epochs: 10	0.732	0.75	0.757
NB	Presence	(1, 2)	alpha: 1.0	0.797	0.814	0.899
SVM	Presence	(1, 2)	C: 10.0, gamma: 0.1, kernel: sigmoid	0.823	0.842	0.797
LogReg	Presence	(1, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.829	0.834	0.905
ANN	Presence	(1, 2)	max_epochs: 12	0.815	0.83	0.88
NB	Frequency	(1, 1)	alpha: 1.0	0.798	0.81	0.879
SVM	Frequency	(1, 1)	C: 10.0, gamma: 0.1, kernel: sigmoid	0.825	0.824	0.797
LogReg	Frequency	(1, 1)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.827	0.842	0.908
ANN	Frequency	(1, 1)	max_epochs: 12	0.821	0.834	0.885
NB	Frequency	(2, 2)	alpha: 1.0	0.741	0.754	0.738
SVM	Frequency	(2, 2)	C: 1.0, gamma: 1.0, kernel: rbf	0.745	0.762	0.578
LogReg	Frequency	(2, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.741	0.762	0.775
ANN	Frequency	(2, 2)	max_epochs: 12	0.728	0.754	0.76
NB	Frequency	(1, 2)	alpha: 0.0001	0.797	0.794	0.862
SVM	Frequency	(1, 2)	C: 10.0, gamma: 0.1, kernel: sigmoid	0.824	0.83	0.802
LogReg	Frequency	(1, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.817	0.836	0.907
ANN	Frequency	(1, 2)	max_epochs: 10	0.817	0.836	0.895
NB	TF-IDF	(1, 1)		0.567	0.5	0.761
SVM	TF-IDF	(1, 1)	C: 10.0, gamma: 0.1, kernel: rbf	0.839	0.848	0.773
LogReg	TF-IDF	(1, 1)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.821	0.828	0.901
ANN	TF-IDF	(1, 1)	max_epochs: 10	0.824	0.84	0.888
NB	TF-IDF	(2, 2)		0.423	0.46	0.706
SVM	TF-IDF	(2, 2)	C: 1.0, kernel: linear	0.743	0.762	0.578
LogReg	TF-IDF	(2, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.744	0.76	0.776
ANN	TF-IDF	(2, 2)	max_epochs: 10	0.732	0.756	0.763
NB	TF-IDF	(1, 2)		0.59	0.5	0.768
SVM	TF-IDF	(1, 2)	C: 10.0, gamma: 0.1, kernel: rbf	0.835	0.844	0.783

LogReg	TF-IDF	(1, 2)	C: 10.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: newton-cg	0.823	0.83	0.901
ANN	TF-IDF	(1, 2)	max_epochs: 12	0.818	0.832	0.891
CNN	Embeddings		max_epochs: 10	0.837	0.842	0.9
LSTM	Embeddings	max_epo 4	0.822	0.842	0.885	
Sentiwordnet					0.453	0.604
Pattern					0.385	0.596
Hu and Liu					0.416	0.624
Vader					0.451	0.631

TABLE A.5: *Detailed case study results for the fifth experimental run.*

Algorithm	Document representation	*n*-gram range	Best parameters	Cross-validated score	Accuracy	AUC
NB	Presence	(1, 1)	alpha: 0.6	0.805	0.804	0.9
SVM	Presence	(1, 1)	C: 10.0, gamma: 0.1, kernel: sigmoid	0.832	0.82	0.784
LogReg	Presence	(1, 1)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.835	0.84	0.91
ANN	Presence	(1, 1)	max_epochs: 12	0.817	0.832	0.907
NB	Presence	(2, 2)	alpha: 0.6	0.742	0.764	0.763
SVM	Presence	(2, 2)	C: 1.0, gamma: 1.0, kernel: rbf	0.744	0.758	0.565
LogReg	Presence	(2, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.745	0.76	0.765
ANN	Presence	(2, 2)	max_epochs: 10	0.738	0.744	0.753
NB	Presence	(1, 2)	alpha: 0.4	0.794	0.802	0.903
SVM	Presence	(1, 2)	C: 10.0, gamma: 0.1, kernel: rbf	0.836	0.834	0.807
LogReg	Presence	(1, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.837	0.84	0.911
ANN	Presence	(1, 2)	max_epochs: 12	0.816	0.84	0.896
NB	Frequency	(1, 1)	alpha: 0.0001	0.8	0.82	0.877
SVM	Frequency	(1, 1)	C: 10.0, gamma: 0.1, kernel: rbf	0.831	0.822	0.795
LogReg	Frequency	(1, 1)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.837	0.828	0.905
ANN	Frequency	(1, 1)	max_epochs: 10	0.816	0.812	0.904
NB	Frequency	(2, 2)	alpha: 0.8	0.742	0.76	0.723
SVM	Frequency	(2, 2)	C: 1.0, gamma: 1.0, kernel: rbf	0.745	0.76	0.566
LogReg	Frequency	(2, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.744	0.76	0.764
ANN	Frequency	(2, 2)	max_epochs: 10	0.731	0.748	0.756
NB	Frequency	(1, 2)	alpha: 0.0001	0.799	0.818	0.884
SVM	Frequency	(1, 2)	C: 10.0, gamma: 0.1, kernel: rbf	0.833	0.826	0.8

LogReg	Frequency	(1, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.833	0.822	0.905
ANN	Frequency	(1, 2)	max_epochs: 12	0.816	0.816	0.898
NB	TF-IDF	(1, 1)		0.558	0.558	0.792
SVM	TF-IDF	(1, 1)	C: 10.0, gamma: 0.1, kernel: rbf	0.842	0.842	0.787
LogReg	TF-IDF	(1, 1)	C: 10.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.828	0.836	0.904
ANN	TF-IDF	(1, 1)	max_epochs: 10	0.822	0.83	0.906
NB	TF-IDF	(2, 2)		0.434	0.444	0.687
SVM	TF-IDF	(2, 2)	C: 1.0, degree: 2, gamma: 1.0, kernel: poly	0.746	0.76	0.569
LogReg	TF-IDF	(2, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.744	0.758	0.77
ANN	TF-IDF	(2, 2)	max_epochs: 12	0.73	0.75	0.757
NB	TF-IDF	(1, 2)		0.58	0.552	0.791
SVM	TF-IDF	(1, 2)	C: 10.0, gamma: 0.1, kernel: rbf	0.846	0.842	0.787
LogReg	TF-IDF	(1, 2)	C: 10.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.832	0.828	0.905
ANN	TF-IDF	(1, 2)	max_epochs: 12	0.809	0.84	0.906
CNN	Embeddings		max_epochs: 12	0.823	0.842	0.911
LSTM	Embeddings		max_epochs: 8	0.824	0.802	0.888
Sentiwordnet					0.475	0.589
Pattern					0.395	0.608
Hu and Liu					0.398	0.601
Vader					0.416	0.588

TABLE A.6: *Detailed case study results for the sixth experimental run.*

Algorithm	Document representation	*n*-gram range	Best parameters	Cross-validated score	Accuracy	AUC
NB	Presence	(1, 1)	alpha: 1.0	0.801	0.808	0.89
SVM	Presence	(1, 1)	C: 10.0, gamma: 0.1, kernel: rbf	0.822	0.828	0.794
LogReg	Presence	(1, 1)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.829	0.832	0.899
ANN	Presence	(1, 1)	max_epochs: 10	0.808	0.826	0.899
NB	Presence	(2, 2)	alpha: 0.0001	0.744	0.76	0.746
SVM	Presence	(2, 2)	C: 1.0, kernel: linear	0.745	0.754	0.573
LogReg	Presence	(2, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.744	0.758	0.742
ANN	Presence	(2, 2)	max_epochs: 10	0.732	0.75	0.736
NB	Presence	(1, 2)	alpha: 0.2	0.791	0.79	0.883
SVM	Presence	(1, 2)	C: 10.0, gamma: 0.1, kernel: sigmoid	0.828	0.83	0.765
LogReg	Presence	(1, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.83	0.834	0.896

ANN	Presence	(1, 2)	max_epochs: 10	0.811	0.838	0.893
NB	Frequency	(1, 1)	alpha: 0.0001	0.796	0.836	0.877
SVM	Frequency	(1, 1)	C: 10.0, gamma: 0.1, kernel: rbf	0.823	0.826	0.79
LogReg	Frequency	(1, 1)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.825	0.822	0.899
ANN	Frequency	(1, 1)	max_epochs: 12	0.811	0.844	0.9
NB	Frequency	(2, 2)	alpha: 1.0	0.741	0.756	0.662
SVM	Frequency	(2, 2)	C: 10.0, gamma: 1.0, kernel: sigmoid	0.746	0.754	0.576
LogReg	Frequency	(2, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.744	0.756	0.735
ANN	Frequency	(2, 2)	max_epochs: 10	0.736	0.75	0.73
NB	Frequency	(1, 2)	alpha: 0.0001	0.795	0.83	0.87
SVM	Frequency	(1, 2)	C: 10.0, gamma: 0.1, kernel: rbf	0.821	0.834	0.785
LogReg	Frequency	(1, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.824	0.824	0.898
ANN	Frequency	(1, 2)	max_epochs: 12	0.804	0.83	0.899
NB	TF-IDF	(1, 1)		0.564	0.496	0.78
SVM	TF-IDF	(1, 1)	C: 1.0, gamma: 1.0, kernel: rbf	0.839	0.832	0.739
LogReg	TF-IDF	(1, 1)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.821	0.826	0.892
ANN	TF-IDF	(1, 1)	max_epochs: 10	0.817	0.842	0.895
NB	TF-IDF	(2, 2)		0.447	0.412	0.673
SVM	TF-IDF	(2, 2)	C: 1.0, degree: 2, gamma: 1.0, kernel: poly	0.746	0.758	0.57
LogReg	TF-IDF	(2, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.744	0.758	0.733
ANN	TF-IDF	(2, 2)	max_epochs: 12	0.74	0.752	0.73
NB	TF-IDF	(1, 2)		0.586	0.506	0.773
SVM	TF-IDF	(1, 2)	C: 10.0, gamma: 0.1, kernel: rbf	0.837	0.848	0.776
LogReg	TF-IDF	(1, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.82	0.826	0.893
ANN	TF-IDF	(1, 2)	max_epochs: 10	0.814	0.834	0.881
CNN	Embeddings		max_epochs: 10	0.829	0.842	0.895
LSTM	Embeddings		max_epochs: 4	0.826	0.83	0.905
Sentiwordnet					0.48	0.619
Pattern					0.424	0.635
Hu and Liu					0.393	0.607
Vader					0.43	0.623

TABLE A.7: *Detailed case study results for the seventh experimental run.*

Algorithm	Document representation	n-gram range	Best parameters	Cross-validated score	Accuracy	AUC
NB	Presence	(1, 1)	alpha: 1.0	0.794	0.834	0.91
SVM	Presence	(1, 1)	C: 10.0, gamma: 0.1, kernel: sigmoid	0.818	0.85	0.804
LogReg	Presence	(1, 1)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.82	0.858	0.908
ANN	Presence	(1, 1)	max_epochs: 10	0.808	0.856	0.902
NB	Presence	(2, 2)	alpha: 0.8	0.748	0.746	0.721
SVM	Presence	(2, 2)	C: 1.0, gamma: 1.0, kernel: rbf	0.75	0.75	0.545
LogReg	Presence	(2, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.749	0.742	0.741
ANN	Presence	(2, 2)	max_epochs: 12	0.732	0.736	0.739
NB	Presence	(1, 2)	alpha: 0.2	0.783	0.826	0.903
SVM	Presence	(1, 2)	C: 10.0, gamma: 0.1, kernel: sigmoid	0.814	0.834	0.785
LogReg	Presence	(1, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.811	0.862	0.909
ANN	Presence	(1, 2)	max_epochs: 10	0.804	0.826	0.902
NB	Frequency	(1, 1)	alpha: 1.0	0.802	0.818	0.876
SVM	Frequency	(1, 1)	C: 10.0, gamma: 0.1, kernel: rbf	0.811	0.844	0.811
LogReg	Frequency	(1, 1)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.814	0.854	0.905
ANN	Frequency	(1, 1)	max_epochs: 12	0.798	0.85	0.899
NB	Frequency	(2, 2)	alpha: 1.0	0.747	0.742	0.673
SVM	Frequency	(2, 2)	C: 1.0, gamma: 1.0, kernel: rbf	0.751	0.75	0.545
LogReg	Frequency	(2, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.746	0.744	0.743
ANN	Frequency	(2, 2)	max_epochs: 10	0.737	0.738	0.733
NB	Frequency	(1, 2)	alpha: 0.0001	0.8	0.806	0.869
SVM	Frequency	(1, 2)	C: 1.0, kernel: linear	0.807	0.83	0.755
LogReg	Frequency	(1, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.81	0.858	0.906
ANN	Frequency	(1, 2)	max_epochs: 10	0.801	0.844	0.901
NB	TF-IDF	(1, 1)		0.58	0.522	0.764
SVM	TF-IDF	(1, 1)	C: 10.0, gamma: 0.1, kernel: rbf	0.83	0.86	0.788
LogReg	TF-IDF	(1, 1)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.816	0.852	0.901
ANN	TF-IDF	(1, 1)	max_epochs: 10	0.805	0.814	0.867
NB	TF-IDF	(2, 2)		0.433	0.44	0.693
SVM	TF-IDF	(2, 2)	C: 1.0, gamma: 1.0, kernel: rbf	0.75	0.748	0.542

LogReg	TF-IDF	(2, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.749	0.748	0.748
ANN	TF-IDF	(2, 2)	max_epochs: 12	0.738	0.742	0.734
NB	TF-IDF	(1, 2)		0.612	0.55	0.779
SVM	TF-IDF	(1, 2)	C: 10.0, gamma: 0.1, kernel: rbf	0.83	0.858	0.791
LogReg	TF-IDF	(1, 2)	C: 10.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.818	0.814	0.883
ANN	TF-IDF	(1, 2)	max_epochs: 12	0.822	0.826	0.889
CNN	Embeddings		max_epochs: 10	0.826	0.854	0.891
LSTM	Embeddings		max_epochs: 4	0.815	0.848	0.898
Sentiwordnet					0.479	0.596
Pattern					0.391	0.607
Hu and Liu					0.424	0.628
Vader					0.447	0.606

TABLE A.8: *Detailed case study results for the eighth experimental run.*

Algorithm	Document representation	*n*-gram range	Best parameters	Cross-validated score	Accuracy	AUC
NB	Presence	(1, 1)	alpha: 1.0	0.815	0.812	0.893
SVM	Presence	(1, 1)	C: 10.0, gamma: 0.1, kernel: rbf	0.828	0.826	0.79
LogReg	Presence	(1, 1)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.823	0.824	0.894
ANN	Presence	(1, 1)	max_epochs: 12	0.814	0.814	0.893
NB	Presence	(2, 2)	alpha: 0.0001	0.748	0.764	0.753
SVM	Presence	(2, 2)	C: 1.0, kernel: linear	0.748	0.75	0.555
LogReg	Presence	(2, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.75	0.756	0.751
ANN	Presence	(2, 2)	max_epochs: 12	0.744	0.756	0.727
NB	Presence	(1, 2)	alpha: 0.2	0.797	0.788	0.894
SVM	Presence	(1, 2)	C: 10.0, gamma: 0.1, kernel: sigmoid	0.823	0.816	0.759
LogReg	Presence	(1, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.815	0.83	0.892
ANN	Presence	(1, 2)	max_epochs: 12	0.812	0.82	0.89
NB	Frequency	(1, 1)	alpha: 0.2	0.807	0.796	0.877
SVM	Frequency	(1, 1)	C: 10.0, gamma: 0.1, kernel: rbf	0.826	0.826	0.788
LogReg	Frequency	(1, 1)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: newton-cg	0.818	0.822	0.894
ANN	Frequency	(1, 1)	max_epochs: 12	0.809	0.826	0.889
NB	Frequency	(2, 2)	alpha: 1.0	0.743	0.75	0.701
SVM	Frequency	(2, 2)	C: 1.0, gamma: 1.0, kernel: rbf	0.75	0.758	0.565
LogReg	Frequency	(2, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.747	0.754	0.747

ANN	Frequency	(2, 2)	max_epochs: 12	0.738	0.754	0.738
NB	Frequency	(1, 2)	alpha: 0.0001	0.812	0.814	0.868
SVM	Frequency	(1, 2)	C: 10.0, gamma: 0.1, kernel: rbf	0.827	0.83	0.787
LogReg	Frequency	(1, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.817	0.826	0.896
ANN	Frequency	(1, 2)	max_epochs: 12	0.807	0.832	0.904
NB	TF-IDF	(1, 1)		0.563	0.534	0.767
SVM	TF-IDF	(1, 1)	C: 10.0, gamma: 0.1, kernel: rbf	0.839	0.838	0.755
LogReg	TF-IDF	(1, 1)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.824	0.838	0.89
ANN	TF-IDF	(1, 1)	max_epochs: 12	0.818	0.826	0.896
NB	TF-IDF	(2, 2)		0.448	0.432	0.661
SVM	TF-IDF	(2, 2)	C: 1.0, degree: 2, gamma: 1.0, kernel: poly	0.75	0.754	0.558
LogReg	TF-IDF	(2, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.748	0.754	0.746
ANN	TF-IDF	(2, 2)	max_epochs: 12	0.736	0.752	0.735
NB	TF-IDF	(1, 2)		0.584	0.566	0.786
SVM	TF-IDF	(1, 2)	C: 1.0, gamma: 1.0, kernel: rbf	0.835	0.844	0.753
LogReg	TF-IDF	(1, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: newton-cg	0.822	0.838	0.89
ANN	TF-IDF	(1, 2)	max_epochs: 10	0.818	0.83	0.885
CNN	Embeddings		max_epochs: 10	0.832	0.842	0.917
LSTM	Embeddings		max_epochs: 4	0.823	0.836	0.918
Sentiwordnet					0.496	0.599
Pattern					0.422	0.639
Hu and Liu					0.41	0.618
Vader					0.447	0.612

TABLE A.9: *Detailed case study results for the ninth experimental run.*

Algorithm	Document representation	*n*-gram range	Best parameters	Cross-validated score	Accuracy	AUC
NB	Presence	(1, 1)	alpha: 0.2	0.807	0.784	0.862
SVM	Presence	(1, 1)	C: 10.0, gamma: 0.1, kernel: rbf	0.823	0.83	0.795
LogReg	Presence	(1, 1)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.834	0.84	0.876
ANN	Presence	(1, 1)	max_epochs: 10	0.825	0.816	0.868
NB	Presence	(2, 2)	alpha: 0.6	0.748	0.76	0.727
SVM	Presence	(2, 2)	C: 1.0, gamma: 1.0, kernel: rbf	0.751	0.754	0.558
LogReg	Presence	(2, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.749	0.752	0.735
ANN	Presence	(2, 2)	max_epochs: 10	0.744	0.746	0.725
NB	Presence	(1, 2)	alpha: 0.2	0.792	0.76	0.859

SVM	Presence	(1, 2)	C: 10.0, gamma: 0.1, kernel: rbf	0.821	0.816	0.789
LogReg	Presence	(1, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.829	0.832	0.879
ANN	Presence	(1, 2)	max_epochs: 12	0.817	0.814	0.868
NB	Frequency	(1, 1)	alpha: 0.2	0.811	0.8	0.848
SVM	Frequency	(1, 1)	C: 10.0, gamma: 0.1, kernel: rbf	0.825	0.828	0.786
LogReg	Frequency	(1, 1)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.826	0.834	0.876
ANN	Frequency	(1, 1)	max_epochs: 12	0.823	0.83	0.869
NB	Frequency	(2, 2)	alpha: 0.8	0.745	0.752	0.665
SVM	Frequency	(2, 2)	C: 1.0, gamma: 1.0, kernel: rbf	0.748	0.754	0.558
LogReg	Frequency	(2, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.746	0.752	0.734
ANN	Frequency	(2, 2)	max_epochs: 12	0.733	0.75	0.733
NB	Frequency	(1, 2)	alpha: 0.0001	0.802	0.8	0.839
SVM	Frequency	(1, 2)	C: 10.0, gamma: 0.1, kernel: rbf	0.824	0.818	0.785
LogReg	Frequency	(1, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.82	0.836	0.876
ANN	Frequency	(1, 2)	max_epochs: 12	0.812	0.816	0.87
NB	TF-IDF	(1, 1)		0.581	0.492	0.737
SVM	TF-IDF	(1, 1)	C: 1.0, gamma: 1.0, kernel: rbf	0.835	0.832	0.746
LogReg	TF-IDF	(1, 1)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.826	0.822	0.862
ANN	TF-IDF	(1, 1)	max_epochs: 12	0.83	0.832	0.868
NB	TF-IDF	(2, 2)		0.45	0.436	0.679
SVM	TF-IDF	(2, 2)	C: 1.0, gamma: 1.0, kernel: rbf	0.748	0.754	0.558
LogReg	TF-IDF	(2, 2)	C: 1.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.745	0.75	0.733
ANN	TF-IDF	(2, 2)	max_epochs: 12	0.732	0.746	0.727
NB	TF-IDF	(1, 2)		0.59	0.524	0.743
SVM	TF-IDF	(1, 2)	C: 1.0, gamma: 1.0, kernel: rbf	0.834	0.826	0.74
LogReg	TF-IDF	(1, 2)	C: 10.0, max_iter: 100, multi_class: auto, n_jobs: 8, solver: sag	0.821	0.822	0.872
ANN	TF-IDF	(1, 2)	max_epochs: 10	0.817	0.816	0.86
CNN	Embeddings		max_epochs: 12	0.839	0.832	0.872
LSTM	Embeddings		max_epochs: 4	0.824	0.822	0.856
Sentiwordnet					0.436	0.557
Pattern					0.402	0.613
Hu and Liu					0.361	0.584
Vader					0.408	0.587

TABLE A.10: *Detailed case study results for the tenth experimental run.*

APPENDIX B

Content analysis for keywords

This appendix contains outputs returned by the ECCO system with respect to the keywords *money, help, staff, consultant, service, ATM* and *time* that were employed in §9.3. More specifically, samples of negative reviews[1] that contain a given keyword are contained in Tables B.1–B.7, whilst word clouds illustrating frequently occurring terms in reviews which mention that particular keyword are shown in Figure B.1.

	Review message
Sample 1	"*I wanted to refund or reverse the money that has deducted from some people ..and ur app ddnt allow me*"
Sample 2	"*Is that I didn't get the help of the money that I was going to borrow & also that u mustn't mix the old debt with the new one because it end up confusing me*"
Sample 3	"*Reversed of money*"
Sample 4	"*I have transferred money to the wrong acc by mistake even now no one is able to help me*"
Sample 5	"*I transfer a money by mistake to the wrong number 2000 u didn't help*"
Sample 6	"*I HAVE UNKOWN DEBIT ORDER I REVESE THE SAME MONEY TWO TIMES BUT STILL MY MONEY DESAPEARING*"

TABLE B.1: *Negative sample reviews that contain the keyword money.*

[1]All sampled reviews have been transcribed exactly as found in the source (spelling and grammatical errors have not been corrected). Furthermore, the name of the industry partner has been replaced by *"_bankName"* in order to preserve anonymity, as it was done during the case study.

(a) *Keyword money* (b) *Keyword help* (c) *Keyword staff* (d) *Keyword consultant*

(e) *Keyword service* (f) *Keyword ATM* (g) *Keyword time*

FIGURE B.1: *Word clouds illustrating frequently occurring terms in negative reviews which contain certain keywords.*

	Review message
Sample 1	"*The lady helped me right at the door,with people going up and down and even after that she did not help*"
Sample 2	"*The consultant that she was helping me, she was not interested by helping, she was slow with low voice, and answering me with shot without any explanation*"
Sample 3	"*Just help when had problem like funerals*"
Sample 4	"*THE LADY WAS HELPNG ME WAS ON A BAD MOOD ANY WAY SORRY FOR THAT BUT EISH.*"
Sample 5	"*The lady who helped me couldn't care less about what I needed, assumed and judged my problem. She had a bad attitude.*"
Sample 6	"*Staff member, that helped me. Was not very accommodating. She let me stand in a long queu at ATM just to verify. That my card was not working. Instead of going outside with me and interjecting the queu.*"

TABLE B.2: *Negative sample reviews that contain the keyword help.*

	Review message
Sample 1	"*Ur staff when the talking if u ask something u dnt nw they so rude they dnt have time actually*"
Sample 2	"*Teach your staff to be more informative about your service*"
Sample 3	"*Just help when had problem like funerals*"
Sample 4	"*Staff at the 1st table need some respect*"
Sample 5	"*You need to train your staff properly in terms of costomer service...And i was looking for settlement letter no one helped me it they said i must leave my details so that they can phone me, even now i haven't receive a call since yesterday.*"
Sample 6	"*Your staff does nt HV good communication skills*"

TABLE B.3: *Negative sample reviews that contain the keyword staff.*

	Review message
Sample 1	"*Consultant did not greet and she was busy talking to her friend on the phone, not welcoming at all*"
Sample 2	"*The consultant was very unprofessional . The one who was standing in front to give the tickets. She didn't understand much . Then the waiting had a confusion . Bad experience .*"
Sample 3	"*Train the consultant more. It seemed that the lady serving us was not sure about the procedure*"
Sample 4	"*Consultant didn't know his job and very slow*"
Sample 5	"*ere are no empty consultant spaces on any of the _bankName branches so as to service clients faster and more effectively.*"
Sample 6	"*The consultant was so rude en give me the number to sit on de line again de other 1 who was nice*"

TABLE B.4: *Negative sample reviews that contain the keyword consultant.*

	Review message
Sample 1	"*If one is used to be welcomed with friendly greetings with a smile, then you got the total opposite from a consultant then the one can't say the service is good*"
Sample 2	"*Da service is too slow*"
Sample 3	"*Generally the service was good, the only area where I feel you can improve is on training new staff better before they start assisting clients. I was assisted by a new staff member and the service was very poor.*"
Sample 4	"*I always get a good service at _bankName but the advise I can give is to advice your consultants to be non judgemental when it comes to the quiries we want them to assist us with,thank you*"
Sample 5	"*poor service no customer care at the branch*"
Sample 6	"*improve the consultant services the lisening skills is poor*"

TABLE B.5: *Negative sample reviews that contain the keyword service.*

	Review message
Sample 1	"*BY CREATING MORE ATM BANK*"
Sample 2	"*LESS CHARGES ON OTHER ATM BANKS CAUSE OF LESS _bankName ATMS*"
Sample 3	"*Avoid long ques at atm*"
Sample 4	"*The speed of the atm*"
Sample 5	"*Install more ATM's 24/7 in Atlantis and surroundings.*"
Sample 6	"*Put security guards on our atm < after office hours*"

TABLE B.6: *Negative sample reviews that contain the keyword ATM.*

	Review message
Sample 1	"*When coin security is busy with putting money we are told they can't help us with deposits and they take over an hour time is precious*"
Sample 2	"*I ddnt receive my money on the time of widrowal and wen I went 2 _bankName they sad I'll receive my money within 30 days*"
Sample 3	"*Most of the time ATM out of order*"
Sample 4	"*I bank with _bankName for more than 9years but yesterday l was not happy with the staff took their time without assisting us people that is a problem*"
Sample 5	"*THE CONSALTEND HLP THE CLINED IN VERRY QWEEK TIME*"
Sample 6	"*The site is not user friendly, and it times out quickly*"

TABLE B.7: *Negative sample reviews that contain the keyword time.*

APPENDIX C

Detailed modelling results for the validation study

The aggregate results achieved during the validation case studies of Chapter 11 are given in Tables C.1–C.3. More specifically, for each data set[1], the algorithm, its input feature set and the hyperparameter values that were most frequently selected by means of the grid search during ten experimental runs are given, along with the frequency with which this particular set of hyperparameters was selected. Furthermore, the mean, minimum and maximum performance achieved in respect of the accuracy metric during testing over ten experimental runs is provided.

Algorithm	Document representation	n-gram range	Best parameters		Accuracy		
			Top	Frequency	Mean	Min	Max
NB	presence	(1, 1)	alpha: 1	7	0.803	0.76	0.84
SVM	presence	(1, 1)	kernel: poly, gamma: 0.01, degree: 2, C: 1	2	0.854	0.83	0.878
LogReg	presence	(1, 1)	solver: sag, n_jobs: 8, multi_class: auto, max_iter: 100, C: 0.1	6	0.87	0.848	0.895
ANN	presence	(1, 1)	val_split: 0.125, solver: Adam, reg_type: None, reg_param: 0.3, max_epochs: 10, loss_function: binary_crossentropy, learning_rate: 0.001, hidden_layers: [100, 100], dropout_prob: 0, decay: 0, batchnorm: False, activation_function: relu	2	0.864	0.842	0.882
NB	presence	(1, 2)	alpha: 1	3	0.826	0.8	0.85
SVM	presence	(1, 2)	kernel: poly, gamma: 1, degree: 1, C: 0.1	2	0.868	0.852	0.89
LogReg	presence	(1, 2)	solver: sag, n_jobs: 8, multi_class: auto, max_iter: 100, C: 10	3	0.88	0.85	0.892

[1]Only the results achieved in respect of the three benchmark data sets, namely the PL04, Yelp and Twitter data sets, are included in this appendix since the results achieved in respect of the SMS data set have already been presented in detail in Appendix A.

ANN	presence	(1, 2)	val_split: 0.125, solver: Adam, reg_type: None, reg_param: 0.2, max_epochs: 30, loss_function: binary_crossentropy, learning_rate: 0.001, hidden_layers: [100, 100], dropout_prob: 0.5, decay: 0, batchnorm: False, activation_function: relu	2	0.879	0.865	0.902
NB	presence	(2, 2)	alpha: 0.8	4	0.812	0.775	0.84
SVM	presence	(2, 2)	kernel: linear, C: 1	2	0.813	0.785	0.85
LogReg	presence	(2, 2)	solver: sag, n_jobs: 8, multi_class: auto, max_iter: 100, C: 1	4	0.826	0.805	0.852
ANN	presence	(2, 2)	val_split: 0.125, solver: Adam, reg_type: None, reg_param: 0.2, max_epochs: 10, loss_function: binary_crossentropy, learning_rate: 0.001, hidden_layers: [100, 100], dropout_prob: 0, decay: 0, batchnorm: False, activation_function: relu	4	0.833	0.805	0.855
NB	frequency	(1, 1)	alpha: 1	7	0.819	0.795	0.845
SVM	frequency	(1, 1)	kernel: poly, gamma: 0.1, degree: 1, C: 1	2	0.834	0.808	0.87
LogReg	frequency	(1, 1)	solver: newton-cg, n_jobs: 8, multi_class: auto, max_iter: 100, C: 0.1	5	0.84	0.818	0.862
ANN	frequency	(1, 1)	val_split: 0.125, solver: Adam, reg_type: None, reg_param: 0.3, max_epochs: 30, loss_function: binary_crossentropy, learning_rate: 0.001, hidden_layers: [100, 100], dropout_prob: 0, decay: 0, batchnorm: False, activation_function: relu	2	0.845	0.69	0.882
NB	frequency	(1, 2)	alpha: 1	7	0.834	0.815	0.852
SVM	frequency	(1, 2)	kernel: poly, gamma: 0.01, degree: 1, C: 0.1	3	0.839	0.768	0.862
LogReg	frequency	(1, 2)	solver: lbfgs, n_jobs: 8, multi_class: auto, max_iter: 100, C: 0.1	4	0.854	0.832	0.878
ANN	frequency	(1, 2)	val_split: 0.125, solver: Adam, reg_type: L2, reg_param: 0.3, max_epochs: 30, loss_function: binary_crossentropy, learning_rate: 0.001, hidden_layers: [100, 100], dropout_prob: 0.5, decay: 0, batchnorm: False, activation_function: relu	2	0.83	0.74	0.875
NB	frequency	(2, 2)	alpha: 1	8	0.829	0.808	0.848
SVM	frequency	(2, 2)	kernel: poly, gamma: 0.01, degree: 1, C: 10	2	0.811	0.782	0.84
LogReg	frequency	(2, 2)	solver: sag, n_jobs: 8, multi_class: auto, max_iter: 100, C: 10	6	0.815	0.802	0.832

ANN	frequency	(2, 2)	val_split: 0.125, solver: Adam, reg_type: None, reg_param: 0.3, max_epochs: 30, loss_function: binary_crossentropy, learning_rate: 0.001, hidden_layers: [100, 100], dropout_prob: 0, decay: 0, batchnorm: False, activation_function: relu	3	0.83	0.765	0.852
NB	tf-idf	(1, 1)			0.659	0.62	0.685
SVM	tf-idf	(1, 1)	kernel: rbf, gamma: 0.1, C: 10	2	0.856	0.82	0.88
LogReg	tf-idf	(1, 1)	solver: sag, n_jobs: 8, multi_class: auto, max_iter: 100, C: 10	3	0.855	0.812	0.882
ANN	tf-idf	(1, 1)	val_split: 0.125, solver: Adam, reg_type: None, reg_param: 0.3, max_epochs: 10, loss_function: binary_crossentropy, learning_rate: 0.001, hidden_layers: [100, 100], dropout_prob: 0.5, decay: 0, batchnorm: False, activation_function: relu	4	0.843	0.82	0.86
NB	tf-idf	(1, 2)			0.709	0.665	0.738
SVM	tf-idf	(1, 2)	kernel: sigmoid, gamma: 1, C: 10	3	0.861	0.838	0.892
LogReg	tf-idf	(1, 2)	solver: sag, n_jobs: 8, multi_class: auto, max_iter: 100, C: 10	3	0.856	0.815	0.89
ANN	tf-idf	(1, 2)	val_split: 0.125, solver: Adam, reg_type: None, reg_param: 0.3, max_epochs: 10, loss_function: binary_crossentropy, learning_rate: 0.001, hidden_layers: [100, 100], dropout_prob: 0.5, decay: 0, batchnorm: False, activation_function: relu	2	0.826	0.5	0.895
NB	tf-idf	(2, 2)			0.732	0.702	0.758
SVM	tf-idf	(2, 2)	kernel: linear, C: 10	2	0.838	0.812	0.868
LogReg	tf-idf	(2, 2)	solver: lbfgs, n_jobs: 8, multi_class: auto, max_iter: 100, C: 10	5	0.839	0.82	0.868
ANN	tf-idf	(2, 2)	val_split: 0.125, solver: Adam, reg_type: None, reg_param: 0.3, max_epochs: 10, loss_function: binary_crossentropy, learning_rate: 0.001, hidden_layers: [100, 100], dropout_prob: 0.5, decay: 0, batchnorm: False, activation_function: relu	3	0.844	0.822	0.865
CNN	Embeddings		val_split: 0.125, solver: Adam, reg_type: L2, reg_param: 0.01, pooling_filter_stride: 2, pooling_filter_size: 2, pooling: yes, max_feat: 46462, max_epochs: 25, loss_function: binary_crossentropy, learning_rate: 0.001, hidden_layers: [[1, 1, 10]], embedding_size: 10, decay: 0, convolution_type: valid, batchnorm: False, activation_function: relu	6	0.764	0.675	0.848

LSTM	Embeddings	val_split: 0.125, solver: Adam, reg_type: L2, reg_param: 0.02, max_feat: 46462, max_epochs: 10, loss_function: binary_crossentropy, learning_rate: 0.001, hidden_layers: [1], embedding_size: 10, dropout_prob: 0, decay: 0	6	0.782	0.67	0.86
Sentiwordnet				0.624	0.605	0.648
Pattern				0.74	0.715	0.762
Hu and Liu				0.701	0.672	0.732
Vader				0.636	0.618	0.67

TABLE C.1: *Aggregate results for the PL04 data set over ten experimental runs.*

Algorithm	Document representation	n-gram range	Best parameters		Accuracy		
			Top	Frequency	Mean	Min	Max
NB	presence	(1, 1)	alpha: 0.4	6	0.899	0.891	0.911
SVM	presence	(1, 1)	kernel: linear, C: 0.1	4	0.919	0.905	0.934
LogReg	presence	(1, 1)	solver: sag, n_jobs: 8, multi_class: auto, max_iter: 100, C: 1	7	0.925	0.912	0.938
ANN	presence	(1, 1)	val_split: 0.125, solver: Adam, reg_type: L2, reg_param: 0.01, max_epochs: 10, loss_function: binary_crossentropy, learning_rate: 0.001, hidden_layers: [10, 10], dropout_prob: 0, decay: 0, batchnorm: False, activation_function: relu	10	0.921	0.894	0.934
NB	presence	(1, 2)	alpha: 0.2	10	0.908	0.89	0.922
SVM	presence	(1, 2)	kernel: linear, C: 0.1	4	0.922	0.909	0.936
LogReg	presence	(1, 2)	solver: sag, n_jobs: 8, multi_class: auto, max_iter: 100, C: 1	5	0.929	0.911	0.94
ANN	presence	(1, 2)	val_split: 0.125, solver: Adam, reg_type: L2, reg_param: 0.01, max_epochs: 10, loss_function: binary_crossentropy, learning_rate: 0.001, hidden_layers: [10, 10], dropout_prob: 0, decay: 0, batchnorm: False, activation_function: relu	10	0.922	0.908	0.94
NB	presence	(2, 2)	alpha: 0.2	8	0.868	0.857	0.886
SVM	presence	(2, 2)	kernel: rbf, gamma: 0.01, C: 10	3	0.85	0.824	0.861
LogReg	presence	(2, 2)	solver: sag, n_jobs: 8, multi_class: auto, max_iter: 100, C: 10	9	0.868	0.845	0.882
ANN	presence	(2, 2)	val_split: 0.125, solver: Adam, reg_type: L2, reg_param: 0.01, max_epochs: 10, loss_function: binary_crossentropy, learning_rate: 0.001, hidden_layers: [10, 10], dropout_prob: 0, decay: 0, batchnorm: False, activation_function: relu	10	0.859	0.84	0.875
NB	frequency	(1, 1)	alpha: 1	5	0.913	0.9	0.924
SVM	frequency	(1, 1)	kernel: rbf, gamma: 0.01, C: 10	3	0.905	0.875	0.937
LogReg	frequency	(1, 1)	solver: sag, n_jobs: 8, multi_class: auto, max_iter: 100, C: 10	5	0.919	0.904	0.933

ANN	frequency	(1, 1)	val_split: 0.125, solver: Adam, reg_type: L2, reg_param: 0.01, max_epochs: 10, loss_function: binary_crossentropy, learning_rate: 0.001, hidden_layers: [10, 10], dropout_prob: 0, decay: 0, batchnorm: False, activation_function: relu	10	0.914	0.894	0.932
NB	frequency	(1, 2)	alpha: 0.2	4	0.917	0.9	0.929
SVM	frequency	(1, 2)	kernel: poly, gamma: 0.1, degree: 1, C: 1	3	0.915	0.904	0.933
LogReg	frequency	(1, 2)	solver: sag, n_jobs: 8, multi_class: auto, max_iter: 100, C: 10	4	0.922	0.908	0.934
ANN	frequency	(1, 2)	val_split: 0.125, solver: Adam, reg_type: L2, reg_param: 0.01, max_epochs: 10, loss_function: binary_crossentropy, learning_rate: 0.001, hidden_layers: [10, 10], dropout_prob: 0, decay: 0, batchnorm: False, activation_function: relu	10	0.914	0.899	0.928
NB	frequency	(2, 2)	alpha: 0.4	6	0.887	0.873	0.899
SVM	frequency	(2, 2)	kernel: poly, gamma: 1, degree: 1, C: 1	2	0.847	0.832	0.862
LogReg	frequency	(2, 2)	solver: sag, n_jobs: 8, multi_class: auto, max_iter: 100, C: 10	10	0.868	0.844	0.883
ANN	frequency	(2, 2)	val_split: 0.125, solver: Adam, reg_type: L2, reg_param: 0.01, max_epochs: 10, loss_function: binary_crossentropy, learning_rate: 0.001, hidden_layers: [10, 10], dropout_prob: 0, decay: 0, batchnorm: False, activation_function: relu	10	0.856	0.838	0.875
NB	tf-idf	(1, 1)			0.687	0.665	0.71
SVM	tf-idf	(1, 1)	kernel: poly, gamma: 0.1, degree: 1, C: 10	3	0.922	0.883	0.942
LogReg	tf-idf	(1, 1)	solver: newton-cg, n_jobs: 8, multi_class: auto, max_iter: 100, C: 10	5	0.926	0.903	0.937
ANN	tf-idf	(1, 1)	val_split: 0.125, solver: Adam, reg_type: L2, reg_param: 0.01, max_epochs: 10, loss_function: binary_crossentropy, learning_rate: 0.001, hidden_layers: [10, 10], dropout_prob: 0, decay: 0, batchnorm: False, activation_function: relu	10	0.915	0.854	0.937
NB	tf-idf	(1, 2)			0.853	0.828	0.869
SVM	tf-idf	(1, 2)	kernel: linear, C: 10	2	0.924	0.896	0.94
LogReg	tf-idf	(1, 2)	solver: sag, n_jobs: 8, multi_class: auto, max_iter: 100, C: 10	5	0.929	0.917	0.938

ANN	tf-idf	(1, 2)	val_split: 0.125, solver: Adam, reg_type: L2, reg_param: 0.01, max_epochs: 10, loss_function: binary_crossentropy, learning_rate: 0.001, hidden_layers: [10, 10], dropout_prob: 0, decay: 0, batchnorm: False, activation_function: relu	10	0.922	0.908	0.937
NB	tf-idf	(2, 2)			0.833	0.807	0.852
SVM	tf-idf	(2, 2)	kernel: sigmoid, gamma: 0.1, C: 10	2	0.867	0.84	0.888
LogReg	tf-idf	(2, 2)	solver: newton-cg, n_jobs: 8, multi_class: auto, max_iter: 100, C: 10	6	0.88	0.854	0.899
ANN	tf-idf	(2, 2)	val_split: 0.125, solver: Adam, reg_type: L2, reg_param: 0.01, max_epochs: 10, loss_function: binary_crossentropy, learning_rate: 0.001, hidden_layers: [10, 10], dropout_prob: 0, decay: 0, batchnorm: False, activation_function: relu	10	0.823	0.627	0.894
CNN	Embeddings		val_split: 0.125, solver: Adam, reg_type: L2, reg_param: 0.001, pooling_filter_stride: 2, pooling_filter_size: 2, pooling: no, max_feat: 8795, max_epochs: 10, loss_function: binary_crossentropy, learning_rate: 0.001, hidden_layers: [[4, 1, 12]], embedding_size: 10, decay: 0, convolution_type: valid, batchnorm: False, activation_function: relu	10	0.915	0.896	0.936
LSTM	Embeddings		val_split: 0.125, solver: Adam, reg_type: None, reg_param: 0, max_feat: 8795, max_epochs: 10, loss_function: binary_crossentropy, learning_rate: 0.01, hidden_layers: [10], embedding_size: 10, dropout_prob: 0, decay: 0, batch_size: 125	10	0.911	0.898	0.929
Sentiwordnet					0.762	0.749	0.782
Pattern					0.868	0.855	0.882
Hu and Liu					0.783	0.769	0.808
Vader					0.825	0.813	0.842

TABLE C.2: *Aggregate results for the Yelp data set over ten experimental runs.*

Algorithm	Document representation	n-gram range	Best parameters		Accuracy		
			Top	Frequency	Mean	Min	Max
NB	presence	(1, 1)	alpha: 0.6	7	0.806	0.79	0.823
SVM	presence	(1, 1)	kernel: poly, gamma: 1, degree: 2, C: 0.1	7	0.8	0.779	0.82
LogReg	presence	(1, 1)	solver: sag, n_jobs: 8, multi_class: auto, max_iter: 100, C: 1	7	0.815	0.802	0.826

ANN	presence	(1, 1)	val_split: 0.125, solver: Adam, reg_type: L2, reg_param: 0.002, max_epochs: 10, loss_function: binary_crossentropy, learning_rate: 0.001, hidden_layers: [50, 50], dropout_prob: 0, decay: 0, batchnorm: False, activation_function: relu	9	0.812	0.796	0.82
NB	presence	(1, 2)	alpha: 0.4	7	0.807	0.794	0.82
SVM	presence	(1, 2)	kernel: rbf, gamma: 0.1, C: 10	7	0.781	0.701	0.804
LogReg	presence	(1, 2)	solver: sag, n_jobs: 8, multi_class: auto, max_iter: 100, C: 1	7	0.811	0.799	0.82
ANN	presence	(1, 2)	val_split: 0.125, solver: Adam, reg_type: L2, reg_param: 0.002, max_epochs: 10, loss_function: binary_crossentropy, learning_rate: 0.001, hidden_layers: [50, 50], dropout_prob: 0, decay: 0, batchnorm: False, activation_function: relu	9	0.805	0.783	0.819
NB	presence	(2, 2)	alpha: 0.2	10	0.703	0.688	0.714
SVM	presence	(2, 2)	kernel: sigmoid, gamma: 0.1, C: 10.0	3	0.701	0.689	0.714
LogReg	presence	(2, 2)	solver: lbfgs, n_jobs: 8, multi_class: auto, max_iter: 100, C: 10.0	3	0.703	0.69	0.713
ANN	presence	(2, 2)	val_split: 0.125, solver: Adam, reg_type: L2, reg_param: 0.002, max_epochs: 10, loss_function: binary_crossentropy, learning_rate: 0.001, hidden_layers: [50, 50], dropout_prob: 0, decay: 0, batchnorm: False, activation_function: relu	9	0.703	0.689	0.715
NB	frequency	(1, 1)	alpha: 0.6	7	0.813	0.804	0.827
SVM	frequency	(1, 1)	kernel: poly, gamma: 1, degree: 1, C: 10	7	0.79	0.693	0.815
LogReg	frequency	(1, 1)	solver: sag, n_jobs: 8, multi_class: auto, max_iter: 100, C: 1	7	0.812	0.802	0.826
ANN	frequency	(1, 1)	val_split: 0.125, solver: Adam, reg_type: L2, reg_param: 0.002, max_epochs: 10, loss_function: binary_crossentropy, learning_rate: 0.001, hidden_layers: [50, 50], dropout_prob: 0, decay: 0, batchnorm: False, activation_function: relu	9	0.813	0.795	0.831
NB	frequency	(1, 2)	alpha: 0.4	7	0.811	0.795	0.823
SVM	frequency	(1, 2)	kernel: poly, gamma: 1, degree: 1, C: 1	7	0.764	0.704	0.815
LogReg	frequency	(1, 2)	solver: sag, n_jobs: 8, multi_class: auto, max_iter: 100, C: 1	7	0.815	0.796	0.833

ANN	frequency	(1, 2)	val_split: 0.125, solver: Adam, reg_type: L2, reg_param: 0.002, max_epochs: 10, loss_function: binary_crossentropy, learning_rate: 0.001, hidden_layers: [50, 50], dropout_prob: 0, decay: 0, batchnorm: False, activation_function: relu	9	0.812	0.795	0.825
NB	frequency	(2, 2)	alpha: 1.0	4	0.707	0.693	0.718
SVM	frequency	(2, 2)	kernel: rbf, gamma: 0.01, C: 10.0	2	0.694	0.682	0.705
LogReg	frequency	(2, 2)	solver: sag, n_jobs: 8, multi_class: auto, max_iter: 100, C: 10.0	5	0.704	0.69	0.713
ANN	frequency	(2, 2)	val_split: 0.125, solver: Adam, reg_type: L2, reg_param: 0.002, max_epochs: 10, loss_function: binary_crossentropy, learning_rate: 0.001, hidden_layers: [50, 50], dropout_prob: 0, decay: 0, batchnorm: False, activation_function: relu	9	0.701	0.69	0.714
NB	tf-idf	(1, 1)			0.593	0.574	0.613
SVM	tf-idf	(1, 1)	kernel: poly, gamma: 1, degree: 2, C: 10	7	0.754	0.693	0.831
LogReg	tf-idf	(1, 1)	solver: newton-cg, n_jobs: 8, multi_class: auto, max_iter: 100, C: 1	7	0.82	0.802	0.831
ANN	tf-idf	(1, 1)	val_split: 0.125, solver: Adam, reg_type: L2, reg_param: 0.002, max_epochs: 10, loss_function: binary_crossentropy, learning_rate: 0.001, hidden_layers: [50, 50], dropout_prob: 0, decay: 0, batchnorm: False, activation_function: relu	9	0.818	0.802	0.827
NB	tf-idf	(1, 2)			0.641	0.629	0.657
SVM	tf-idf	(1, 2)	kernel: rbf, gamma: 0.1, C: 10	7	0.784	0.67	0.824
LogReg	tf-idf	(1, 2)	solver: sag, n_jobs: 8, multi_class: auto, max_iter: 100, C: 1	7	0.819	0.805	0.83
ANN	tf-idf	(1, 2)	val_split: 0.125, solver: Adam, reg_type: L2, reg_param: 0.002, max_epochs: 10, loss_function: binary_crossentropy, learning_rate: 0.001, hidden_layers: [50, 50], dropout_prob: 0, decay: 0, batchnorm: False, activation_function: relu	9	0.815	0.807	0.829
NB	tf-idf	(2, 2)			0.381	0.356	0.4
SVM	tf-idf	(2, 2)	kernel: rbf, gamma: 0.1, C: 10.0	4	0.698	0.67	0.719
LogReg	tf-idf	(2, 2)	solver: newton-cg, n_jobs: 8, multi_class: auto, max_iter: 100, C: 10.0	3	0.705	0.694	0.717

ANN	tf-idf	(2, 2)	val_split: 0.125, solver: Adam, reg_type: L2, reg_param: 0.002, max_epochs: 10, loss_function: binary_crossentropy, learning_rate: 0.001, hidden_layers: [50, 50], dropout_prob: 0, model__decay: 0, model__batchnorm: False, model__activation_function: relu	9	0.701	0.677	0.718
CNN	Embeddings		val_split: 0.125, solver: Adam, reg_type: L2, reg_param: 0.02, pooling_filter_stride: 2, pooling_filter_size: 2, pooling: yes, max_feat: 8462, max_epochs: 10, loss_function: binary_crossentropy, learning_rate: 0.01, hidden_layers: [[3, 1, 12]], embedding_size: 10, decay: 0, convolution_type: valid, batchnorm: False, activation_function: relu	9	0.778	0.755	0.801
LSTM	Embeddings		val_split: 0.125, solver: Adam, reg_type: None, reg_param: 0, max_feat: 8462, max_epochs: 5, loss_function: binary_crossentropy, learning_rate: 0.03, hidden_layers: [3], embedding_size: 10, dropout_prob: 0, decay: 0, batch_size: 125	9	0.787	0.774	0.804
Sentiwordnet				7	0.494	0.463	0.517
Pattern				7	0.625	0.599	0.642
Hu and Liu				7	0.643	0.618	0.665
Vader				7	0.861	0.854	0.868

TABLE C.3: *Aggregate results for the Twitter data set over ten experimental runs.*

APPENDIX D

Detailed modelling results for the ensemble methods

This appendix contains the results of the ensemble modelling phase (Process 12.0 of the ECCO framework) for the case study in Chapter 9 and the validation studies in Chapter 11. Tables D.1, D.3, D.5 and D.7 contain the details of the base learners selected for the PL04, Yelp, Twitter and SMS data sets, respectively[1]. Tables D.2, D.4, D.6 and D.8 contain the aggregate results achieved by the eighteen configurations (2 output types $\times$ 3 combination methods $\times$ 3 selection approaches) of ensembles formed using subsets of these base learners in respect of the same data sets.

[1]The aggregate results of the base learners reported in these tables may differ slightly from those given in Appendices A and C since a variation of a random search was employed in place of a grid search during hyperparameter tuning if the search space of possible hyperparameter values was larger than 10, as explained in §11.2.

Base learner	Algorithm	Document representation	n-gram range	Accuracy			AUC		
				Min	Median	Max	Min	Median	Max
0	LogReg	presence	(1, 1)	0.848	0.871	0.895	0.929	0.9435	0.955
1	ANN	presence	(1, 1)	0.828	0.876	0.892	0.937	0.945	0.951
2	SVM	presence	(1, 2)	0.852	0.867	0.878	0.935	0.939	0.953
3	LogReg	presence	(1, 2)	0.848	0.879	0.895	0.939	0.9475	0.958
4	ANN	presence	(1, 2)	0.83	0.863	0.895	0.935	0.9445	0.959
5	SVM	tf-idf	(1, 1)	0.762	0.837	0.875	0.829	0.924	0.942
6	Pattern			0.718	0.742	0.762	0.718	0.742	0.762

TABLE D.1: *Details of the base learners for the PL04 data set.*

Base learners	Output type	Combination method	Selection approach	Accuracy			AUC		
				Min	Median	Max	Min	Median	Max
0, 1, 2, 3, 4	Discrete	Simple	Greedy	0.86	0.889	0.9	0.86	0.889	0.9
0, 1, 2, 3, 4	Discrete	Weighted	Greedy	0.86	0.888	0.9	0.86	0.888	0.9
0, 1, 2, 3, 4	Discrete	Meta	Greedy	0.868	0.894	0.912	0.92	0.9295	0.948
0, 1, 2, 3, 4	Score	Simple	Greedy	0.865	0.893	0.912	0.946	0.958	0.966
0, 1, 2, 3, 4	Score	Weighted	Greedy	0.865	0.893	0.91	0.944	0.956	0.967
0, 1, 2, 3, 4	Score	Meta	Greedy	0.87	0.895	0.91	0.944	0.956	0.967
3, 6	Discrete	Simple	Model	0.778	0.795	0.82	0.777	0.795	0.82
3, 6	Discrete	Weighted	Model	0.85	0.879	0.895	0.85	0.879	0.895
3, 6	Discrete	Meta	Model	0.858	0.878	0.892	0.888	0.911	0.925
3, 6	Score	Simple	Model	0.718	0.742	0.762	0.889	0.911	0.921
3, 6	Score	Weighted	Model	0.795	0.836	0.87	0.908	0.926	0.936
3, 6	Score	Meta	Model	0.868	0.885	0.898	0.939	0.951	0.959
3, 5	Discrete	Simple	Model_ML	0.835	0.86	0.89	0.835	0.86	0.89
3, 5	Discrete	Weighted	Model_ML	0.852	0.876	0.898	0.852	0.876	0.898
3, 5	Discrete	Meta	Model_ML	0.85	0.88	0.895	0.889	0.91	0.935
3, 5	Score	Simple	Model_ML	0.855	0.878	0.905	0.93	0.952	0.962
3, 5	Score	Weighted	Model_ML	0.852	0.88	0.905	0.93	0.9525	0.962
3, 5	Score	Meta	Model_ML	0.855	0.889	0.902	0.936	0.954	0.964

TABLE D.2: *Aggregate results of the ensemble learners for the PL04 data set over ten experimental runs. Base learners are referred to in terms of the indices from Table D.1.*

Base learner	Algorithm	Document representation	n-gram range	Accuracy			AUC		
				Min	Median	Max	Min	Median	Max
0	LogReg	presence	(1, 1)	0.911	0.925	0.94	0.97	0.975	0.983
1	LogReg	presence	(1, 2)	0.909	0.931	0.94	0.973	0.978	0.987
2	SVM	tf-idf	(1, 1)	0.911	0.93	0.941	0.971	0.981	0.985
3	LogReg	tf-idf	(1, 1)	0.903	0.93	0.937	0.971	0.98	0.985
4	SVM	tf-idf	(1, 2)	0.603	0.922	0.942	0.945	0.98	0.987
5	LogReg	tf-idf	(1, 2)	0.917	0.93	0.938	0.974	0.982	0.987
6	Sentiwordnet			0.853	0.864	0.88	0.845	0.86	0.873

TABLE D.3: *Details of the base learners for the Yelp data set.*

Base learners	Output type	Combination method	Selection approach	Accuracy			AUC		
				Min	Median	Max	Min	Median	Max
0, 1, 3, 4, 5	Discrete	Simple	Greedy	0.919	0.931	0.94	0.917	0.928	0.938
0, 1, 3, 4, 5	Discrete	Weighted	Greedy	0.919	0.932	0.94	0.917	0.93	0.938
0, 1, 3, 4, 5	Discrete	Meta	Greedy	0.919	0.932	0.937	0.939	0.955	0.966
0, 1, 3, 4, 5	Score	Simple	Greedy	0.917	0.932	0.945	0.976	0.982	0.989
0, 1, 3, 4, 5	Score	Weighted	Greedy	0.917	0.932	0.943	0.976	0.982	0.989
0, 1, 3, 4, 5	Score	Meta	Greedy	0.924	0.931	0.95	0.976	0.983	0.988
1, 6	Discrete	Simple	Model	0.891	0.906	0.919	0.897	0.912	0.924
1, 6	Discrete	Weighted	Model	0.909	0.931	0.938	0.908	0.929	0.937
1, 6	Discrete	Meta	Model	0.909	0.932	0.94	0.942	0.956	0.966
1, 6	Score	Simple	Model	0.853	0.864	0.88	0.964	0.972	0.977
1, 6	Score	Weighted	Model	0.899	0.908	0.926	0.968	0.976	0.98
1, 6	Score	Meta	Model	0.92	0.931	0.947	0.974	0.981	0.987
1, 2	Discrete	Simple	Model_ML	0.913	0.928	0.942	0.917	0.929	0.944
1, 2	Discrete	Weighted	Model_ML	0.911	0.93	0.941	0.908	0.93	0.938
1, 2	Discrete	Meta	Model_ML	0.909	0.928	0.941	0.933	0.943	0.96
1, 2	Score	Simple	Model_ML	0.922	0.934	0.947	0.974	0.982	0.988
1, 2	Score	Weighted	Model_ML	0.924	0.935	0.946	0.974	0.982	0.988
1, 2	Score	Meta	Model_ML	0.921	0.934	0.946	0.974	0.982	0.988

TABLE D.4: *Aggregate results of the ensemble learners for the Yelp data set over ten experimental runs. Base learners are referred to in terms of the indices from Table D.3.*

Base learner	Algorithm	Document representation	n-gram range	Accuracy			AUC		
				Min	Median	Max	Min	Median	Max
0	ANN	frequency	(1, 2)	0.801	0.808	0.826	0.874	0.889	0.904
1	LogReg	tf-idf	(1, 1)	0.788	0.802	0.814	0.887	0.904	0.915
2	ANN	tf-idf	(1, 1)	0.807	0.815	0.829	0.874	0.893	0.902
3	LogReg	tf-idf	(1, 2)	0.787	0.811	0.829	0.884	0.9	0.905
4	ANN	tf-idf	(1, 2)	0.806	0.812	0.824	0.876	0.889	0.899
5	Vader			0.85	0.859	0.865	0.885	0.891	0.896

TABLE D.5: *Details of the base learners for the Twitter data set.*

Base learners	Output type	Combination method	Selection approach	Accuracy			AUC		
				Min	Median	Max	Min	Median	Max
1, 2, 3, 4, 5	Discrete	Simple	Greedy	0.817	0.827	0.84	0.755	0.774	0.786
1, 2, 3, 4, 5	Discrete	Weighted	Greedy	0.819	0.826	0.842	0.757	0.766	0.79
1, 2, 3, 4, 5	Discrete	Meta	Greedy	0.898	0.906	0.915	0.932	0.94	0.947
1, 2, 3, 4, 5	Score	Simple	Greedy	0.843	0.854	0.863	0.933	0.938	0.945
1, 2, 3, 4, 5	Score	Weighted	Greedy	0.843	0.851	0.861	0.931	0.938	0.944
1, 2, 3, 4, 5	Score	Meta	Greedy	0.899	0.906	0.913	0.949	0.956	0.965
1, 5	Discrete	Simple	Model	0.84	0.853	0.858	0.884	0.891	0.896
1, 5	Discrete	Weighted	Model	0.788	0.802	0.814	0.705	0.721	0.742
1, 5	Discrete	Meta	Model	0.9	0.906	0.914	0.928	0.936	0.943
1, 5	Score	Simple	Model	0.85	0.859	0.865	0.942	0.948	0.954
1, 5	Score	Weighted	Model	0.85	0.861	0.868	0.942	0.948	0.954
1, 5	Score	Meta	Model	0.9	0.905	0.913	0.95	0.955	0.965
0, 1	Discrete	Simple	Model_ML	0.8	0.816	0.83	0.759	0.778	0.802
0, 1	Discrete	Weighted	Model_ML	0.788	0.802	0.814	0.705	0.721	0.742
0, 1	Discrete	Meta	Model_ML	0.801	0.817	0.833	0.764	0.794	0.813
0, 1	Score	Simple	Model_ML	0.801	0.815	0.829	0.889	0.899	0.911
0, 1	Score	Weighted	Model_ML	0.8	0.822	0.833	0.886	0.9	0.914
0, 1	Score	Meta	Model_ML	0.806	0.822	0.839	0.888	0.905	0.915

TABLE D.6: *Aggregate results of the ensemble learners for the Twitter data set over ten experimental runs. Base learners are referred to in terms of the indices from Table D.5.*

Base learner	Algorithm	Document representation	n-gram range	Accuracy			AUC		
				Min	Median	Max	Min	Median	Max
0	LogReg	presence	(1, 1)	0.814	0.84	0.854	0.887	0.9	0.926
1	LogReg	presence	(1, 2)	0.816	0.836	0.859	0.886	0.899	0.92
2	LogReg	frequency	(1, 1)	0.799	0.839	0.859	0.879	0.9	0.923
3	ANN	frequency	(1, 1)	0.795	0.822	0.848	0.866	0.89	0.912
4	LogReg	frequency	(1, 2)	0.812	0.831	0.857	0.887	0.895	0.924
5	LogReg	tf-idf	(1, 1)	0.818	0.828	0.85	0.88	0.893	0.925
6	CNN	Embeddings		0.789	0.826	0.852	0.867	0.882	0.907
7	Sentiwordnet			0.436	0.47	0.496	0.557	0.592	0.619

TABLE D.7: *Details of the base learners for the SMS data set.*

Base learners	Output type	Combination method	Selection approach	Accuracy			AUC		
				Min	Median	Max	Min	Median	Max
0, 1, 2, 4, 5	Discrete	Simple	Greedy	0.818	0.843	0.861	0.774	0.798	0.814
0, 1, 2, 4, 5	Discrete	Weighted	Greedy	0.818	0.843	0.861	0.774	0.798	0.814
0, 1, 2, 4, 5	Discrete	Meta	Greedy	0.801	0.827	0.85	0.794	0.819	0.843
0, 1, 2, 4, 5	Score	Simple	Greedy	0.816	0.836	0.865	0.892	0.902	0.927
0, 1, 2, 4, 5	Score	Weighted	Greedy	0.816	0.837	0.865	0.892	0.902	0.927
0, 1, 2, 4, 5	Score	Meta	Greedy	0.83	0.844	0.865	0.89	0.898	0.924
1, 6, 7	Discrete	Simple	Model	0.818	0.841	0.859	0.77	0.79	0.835
1, 6, 7	Discrete	Weighted	Model	0.811	0.832	0.852	0.769	0.796	0.823
1, 6, 7	Discrete	Meta	Model	0.809	0.825	0.85	0.827	0.853	0.886
1, 6, 7	Score	Simple	Model	0.779	0.803	0.822	0.86	0.874	0.885
1, 6, 7	Score	Weighted	Model	0.816	0.837	0.855	0.878	0.894	0.912
1, 6, 7	Score	Meta	Model	0.82	0.849	0.865	0.894	0.904	0.927
1, 3, 6	Discrete	Simple	Model_ML	0.82	0.846	0.867	0.761	0.792	0.813
1, 3, 6	Discrete	Weighted	Model_ML	0.814	0.84	0.867	0.753	0.794	0.816
1, 3, 6	Discrete	Meta	Model_ML	0.818	0.834	0.859	0.804	0.839	0.868
1, 3, 6	Score	Simple	Model_ML	0.812	0.854	0.863	0.887	0.903	0.926
1, 3, 6	Score	Weighted	Model_ML	0.818	0.849	0.873	0.894	0.902	0.922
1, 3, 6	Score	Meta	Model_ML	0.814	0.844	0.863	0.891	0.903	0.922

TABLE D.8: *Aggregate results of the ensemble learners for the SMS data set over ten experimental runs. Base learners are given in terms of the indices from Table D.7.*